# Enhancing B language reasoners with SMT techniques

**Vincent Trélat**

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

# Outline

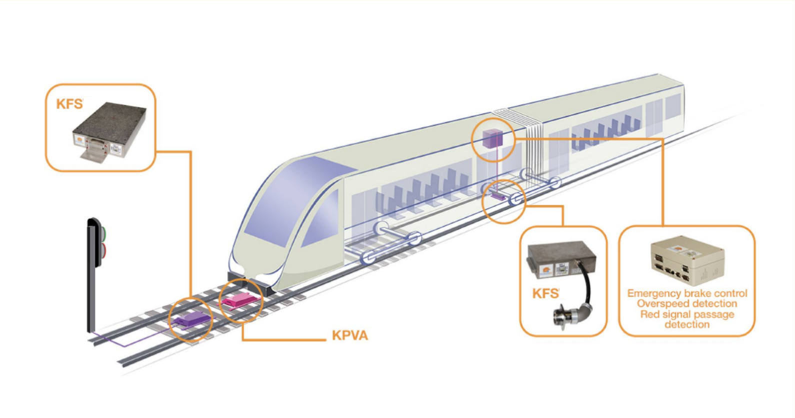
1. **Introduction**
   - Ph.D. subject
   - B and Atelier B
   - SMT-LIB
2. **Encoding B proof obligations in SMT-LIB using HOL**
   - Encoding sets
   - Encoding functions
   - Example
3. **Conclusion**

# Context

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

# Context

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



**Subject**: Enhancing B language reasoners with SMT techniques

**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)))))))
```

**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)))))))
```

**Subject**: Enhancing B language reasoners with SMT techniques

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

$\longrightarrow$

```
(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 (FO) predicate prover
- Tries to automatically discharge proof obligations

## SMT-LIB

- Standard input 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)

1. Introduction
   - Ph.D. subject
   - B and Atelier B
   - SMT-LIB
2. **Encoding B proof obligations in SMT-LIB using HOL**
   - Encoding sets
   - Encoding functions
   - Example
3. Conclusion

## 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 ($\lambda$-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 ($\lambda$-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 ($\lambda$-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 ($\lambda$-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 ($\lambda$-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 $f : A \nrightarrow B$ 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**

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

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

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

$$\vdots$$

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

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\,\tau \times \iota \cdot f \in S \rightarrowtail a..b$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\,\tau \times \iota \cdot f \in S \rightarrowtail a..b$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\,\tau \times \iota \cdot f \in S \twoheadrightarrow a..b \land S \subseteq \mathbf{dom}(f) \land \mathtt{\_inj}(f)$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\,\tau \times \iota \cdot f \in S \nrightarrow a..b \land S \subseteq \mathbf{dom}(f) \land \_\mathtt{inj}(f)$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\,\tau \times \iota \cdot f \in S \leftrightarrow a..b \land \mathtt{\_func}(f) \land S \subseteq \mathbf{dom}(f) \land \mathtt{\_inj}(f)$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\,\tau \times \iota \cdot f \in S \leftrightarrow a..b \land \_\mathtt{func}(f) \land S \subseteq \mathbf{dom}(f) \land \_\mathtt{inj}(f)$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{array}{ll}
\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\, \tau \times \iota \cdot & \\
\qquad f \in S \leftrightarrow a..b & \wedge \\
\qquad \forall x : \tau, y : \tau, z : \iota \cdot x \mapsto y \in f \wedge x \mapsto z \in f \Rightarrow y = z & \wedge \\
\qquad S \subseteq \mathbf{dom}(f) & \wedge \\
\qquad \mathtt{\_inj}(f) &
\end{array}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{aligned}
\forall a : \iota \cdot \exists b : \iota, f : \texttt{set}\,\tau \times \iota \cdot \\
& f \in S \leftrightarrow a..b && \wedge \\
& \forall x : \tau, y : \iota, z : \iota \cdot x \mapsto y \in f \wedge x \mapsto z \in f \Rightarrow y = z && \wedge \\
& S \subseteq \mathbf{dom}(f) && \wedge \\
& \texttt{\_inj}(f)
\end{aligned}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{aligned}
&\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\, \tau \times \iota \cdot \\
&\qquad \forall x : \tau, y : \iota \cdot x \mapsto y \in f \Rightarrow x \in S \land a \leq y \land y \leq b \qquad \land \\
&\qquad \forall x : \tau, y : \tau, z : \iota \cdot x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z \qquad \land \\
&\qquad S \subseteq \mathbf{dom}(f) \qquad \land \\
&\qquad \_\mathtt{inj}(f)
\end{aligned}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{array}{ll}
\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\,\tau \times \iota \cdot & \\
\quad \forall x : \tau, y : \iota \cdot x \mapsto y \in f \Rightarrow x \in S \land a \leq y \land y \leq b & \land \\
\quad \forall x : \tau, y : \iota, z : \iota \cdot x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z & \land \\
\quad S \subseteq \mathbf{dom}(f) & \land \\
\quad \mathtt{\_inj}(f) &
\end{array}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{aligned}
&\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\, \tau \times \iota \cdot \\
&\qquad \forall x : \tau, y : \iota \cdot x \mapsto y \in f \Rightarrow x \in S \land a \leq y \land y \leq b && \land \\
&\qquad \forall x : \tau, y : \iota, z : \iota \cdot x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z && \land \\
&\qquad \forall z : \tau \cdot z \in S \Rightarrow \exists w : \iota \cdot z \mapsto w \in f && \land \\
&\qquad \mathtt{\_inj}(f)
\end{aligned}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{aligned}
&\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\, \tau \times \iota \cdot \\
&\qquad \forall x : \tau, y : \iota \cdot x \mapsto y \in f \Rightarrow x \in S \land a \leq y \land y \leq b && \land \\
&\qquad \forall x : \tau, y : \iota, z : \iota \cdot x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z && \land \\
&\qquad \forall z : \tau \cdot z \in S \Rightarrow \exists w : \iota \cdot z \mapsto w \in f && \land \\
&\qquad \mathtt{\_inj}(f)
\end{aligned}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{aligned}
&\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\, \tau \times \iota \cdot \\
&\qquad \forall x : \tau, y : \iota \cdot x \mapsto y \in f \Rightarrow x \in S \land a \leq y \land y \leq b && \land \\
&\qquad \forall x : \tau, y : \iota, z : \iota \cdot x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z && \land \\
&\qquad \forall z : \tau \cdot z \in S \Rightarrow \exists w : \iota \cdot z \mapsto w \in f && \land \\
&\qquad \forall x : \tau, y : \tau, z : \tau \cdot x \mapsto z \in f \land y \mapsto z \in f \Rightarrow x = y
\end{aligned}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{aligned}
\forall a : \iota \cdot \exists b : \iota, f : \mathtt{set}\ \tau \times \iota \cdot \quad & \\
& \forall x : \tau, y : \iota \cdot x \mapsto y \in f \Rightarrow x \in S \land a \leq y \land y \leq b && \land \\
& \forall x : \tau, y : \iota, z : \iota \cdot x \mapsto y \in f \land x \mapsto z \in f \Rightarrow y = z && \land \\
& \forall z : \tau \cdot z \in S \Rightarrow \exists w : \iota \cdot z \mapsto w \in f && \land \\
& \forall x : \tau, y : \tau, z : \tau \cdot x \mapsto z \in f \land y \mapsto z \in f \Rightarrow x = y
\end{aligned}
$$

# How large is the overhead?

Let $S$ be a set of elements of type $\tau$. The expression **finite** $S$ is translated as follows:

$$
\begin{aligned}
\exists\, N : \iota, f : \tau \to \iota \cdot\ & \\
& \forall x : \tau, y : \tau, z : \iota \cdot f(x) = z \land f(y) = z \Rightarrow x = y \qquad \land \\
& \forall x : \tau \cdot x \in S \Rightarrow f(x) < N
\end{aligned}
$$

# A non-working example

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

Let $f \in \mathbb{Z} \rightarrow\!\!\!\!+ \mathbb{Z}$. 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

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

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

# A non-working example

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)

# A non-working example

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$(0, 2)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 0 \mapsto 2\}$

# A non-working example

Let $f \in \mathbb{Z} \pto \mathbb{Z}$. 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 $\mathrm{op}(2,3)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 3\}$ ✓ (still a function)
- After $\mathrm{op}(0,2)$, $f = \{0 \mapsto 1, 1 \mapsto 0, 0 \mapsto 2\}$ ✗ (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:

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

Idea of the encoding for "true" functions:

## Rule: partial function

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

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

$\cdot^?$ 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:

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

$\cdot^?$ 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. $\uparrow\mathbb{N} = \uparrow\{e_i\}_{i \in \mathcal{I}} = \mathbb{Z}$)

# Example

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

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

# Example

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

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

# Example

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

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

```
(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

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

```
(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

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

```
(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

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

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

```
(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

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

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

```
(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)))))
   ))
```

# 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

## Questions?