BARREL

A modern backend for Atelier B in Lean

Vincent Trélat, Ghilain Bergeron

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

$\exists x.(x \in \mathbb{Z} \wedge x \notin \mathbb{Z})$

```
--------------
```

$\exists x.(x \in \mathbb{Z} \wedge x \notin \mathbb{Z})$

```
se(min(∅))
```

```
--------------
 ∃x.(x ∈ ℤ ∧ x ∉ ℤ)
 se(min(∅))
--------------
 min(∅) ∈ ℤ ∧ min(∅) ∉ ℤ
```

```
--------------
 ∃x.(x ∈ ℤ ∧ x ∉ ℤ)
 se(min(∅))
--------------
 min(∅) ∈ ℤ ∧ min(∅) ∉ ℤ
 mp
```

```
--------------
 ∃x.(x ∈ ℤ ∧ x ∉ ℤ)
 se(min(∅))
--------------
 min(∅) ∈ ℤ ∧ min(∅) ∉ ℤ
 mp
--------------
"Assertion is proved."
```

What just happened?

- $\min(S)$ is **partial**: defined only when $S \subseteq \mathbb{Z}$ is non-empty and bounded below.

What just happened?

- $\min(S)$ is **partial**: defined only when $S \subseteq \mathbb{Z}$ is non-empty and bounded below.
- Atelier B does generate WD obligations...

What just happened?

- $\min(S)$ is **partial**: defined only when $S \subseteq \mathbb{Z}$ is non-empty and bounded below.
- Atelier B does generate WD obligations...
- ...but they are disconnected from the goal;

What just happened?

- $\min(S)$ is **partial**: defined only when $S \subseteq \mathbb{Z}$ is non-empty and bounded below.
- Atelier B does generate WD obligations…
- …but they are disconnected from the goal;
- …and tactics (e.g. `se`, `ah`) don't enforce them on instantiation.

What just happened?

- $\min(S)$ is **partial**: defined only when $S \subseteq \mathbb{Z}$ is non-empty and bounded below.
- Atelier B does generate WD obligations…
- …but they are disconnected from the goal;
- …and tactics (e.g. `se`, `ah`) don't enforce them on instantiation.

Result: ill-defined terms can sneak into POs,

What just happened?

- $\min(S)$ is **partial**: defined only when $S \subseteq \mathbb{Z}$ is non-empty and bounded below.
- Atelier B does generate WD obligations…
- …but they are disconnected from the goal;
- …and tactics (e.g. `se`, `ah`) don't enforce them on instantiation.

Result: ill-defined terms can sneak into POs, and any goal can be discharged using:

$$\texttt{ah}(\exists x.(x \in \mathbb{Z} \land x \notin \mathbb{Z}));\texttt{se}(\texttt{min}(\varnothing));\texttt{mp};\texttt{mp}$$

What just happened?

- $\min(S)$ is **partial**: defined only when $S \subseteq \mathbb{Z}$ is non-empty and bounded below.
- Atelier B does generate WD obligations…
- …but they are disconnected from the goal;
- …and tactics (e.g. `se`, `ah`) don't enforce them on instantiation.

Result: ill-defined terms can sneak into POs, and any goal can be discharged using:

$$\texttt{ah}(\exists x.(x \in \mathbb{Z} \land x \notin \mathbb{Z}));\texttt{se}(\texttt{min}(\varnothing));\texttt{mp};\texttt{mp}$$

We can make it safer with a modern proof assistant.

Context

The B method and Atelier B

- **The B method**: formal software development via refinement of state machines.
- **Atelier B**: industrial tool for B, including
  - a PO generator,
  - internal/external (automated) provers,
  - an interactive prover.

Context

Why a modern backend?

- Atelier B's prover is **trust-by-tradition** (and closed).
- Existing alternatives are partial:
  - Isabelle/HOL for Event-B (Rodin plugin, DSL)
  - Rocq for B semantics (no integrated backend)
  - SMT-based discharge of POs (BEer, ppTrans)

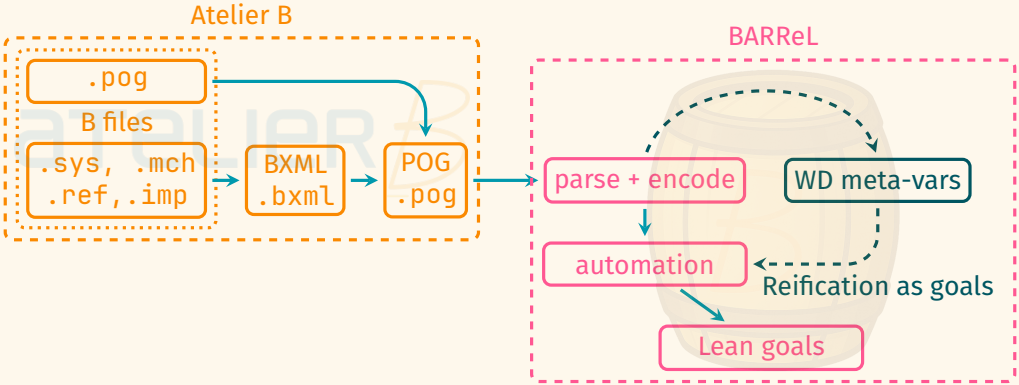
Context

Why a modern backend?

- Atelier B's prover is **trust-by-tradition** (and closed).
- Existing alternatives are partial:
  - Isabelle/HOL for Event-B (Rodin plugin, DSL)
  - Rocq for B semantics (no integrated backend)
  - SMT-based discharge of POs (BEer, ppTrans)
- We want:
  - interactive proofs on the original B POs (or close enough),
  - a rich mathematical library (*e. g.*, Mathlib),
  - and by-construction treatment of WD.

B Automated tRanslation for Reasoning in Lean

BARReL
Pipeline



Encoding B in Lean

Sets, relations, functions

- B notations are reproduced verbatim: $\subseteq$, $\in$, $\cup$, $\cap$, $\mathcal{P}$, FIN, FIN_1, $\rightarrow$, $\nrightarrow$, $\rightarrowtail$, ...

Encoding B in Lean

Sets, relations, functions

- B notations are reproduced verbatim: $\subseteq$, $\in$, $\cup$, $\cap$, $\mathcal{P}$, FIN, FIN_1, $\rightarrow$, $\nrightarrow$, $\nrightarrow$, ...
- Encoding: straightforward set-based mapping $\mathcal{B} \rightarrow$ LEAN using Mathlib.

Encoding B in Lean

Sets, relations, functions

- B notations are reproduced verbatim: $\subseteq$, $\in$, $\cup$, $\cap$, $\mathcal{P}$, FIN, FIN_1, $\rightarrow$, $\nrightarrow$, $\rightarrowtail$, ...
- Encoding: straightforward set-based mapping $\mathcal{B} \rightarrow$ LEAN using Mathlib.
- Coverage: logic, sets, relations, functions, arithmetic, sequences, ...

Encoding B in Lean

Partial operators

Many B operators are partial: min, max, card, $F(x)$, ...

Encoding B in Lean

Partial operators

Many B operators are partial: min, max, card, $F(x)$, ...
BARREL encodes them as total Lean functions taking proofs of WD as arguments:

Encoding B in Lean

Partial operators

Many B operators are partial: min, max, card, $F(x)$, ...
BARREL encodes them as total Lean functions taking proofs of WD as arguments:

```
structure min.WD [PartialOrder α] (S : Set α)
  where
    isBoundedBelow :
      ∃ x ∈ S, ∀ y ∈ S, x ≤ y

def min (S : Set α) (wd : min.WD S) : α :=
  Classical.choose wd.isBoundedBelow
```

```
structure app.WD (f : SetRel α β) (x : α)
  where
    isPartialFunction : f ∈ dom f → ran f
    isInDomain        : x ∈ dom f

def app (f : SetRel α β) (x : α)
  (wd : app.WD f x) : β :=
    Classical.choose wd.isInDomain
```

Encoding B in Lean

Partial operators

Many B operators are partial: min, max, card, $F(x)$, ...
BARREL encodes them as total Lean functions taking proofs of WD as arguments:

```
structure min.WD [PartialOrder α] (S : Set α)
  where
    isBoundedBelow :
      ∃ x ∈ S, ∀ y ∈ S, x ≤ y

def min (S : Set α) (wd : min.WD S) : α :=
  Classical.choose wd.isBoundedBelow
```

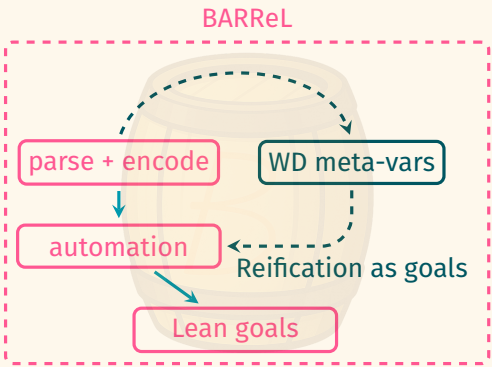
```
structure app.WD (f : SetRel α β) (x : α)
  where
    isPartialFunction : f ∈ dom f → ran f
    isInDomain        : x ∈ dom f

def app (f : SetRel α β) (x : α)
  (wd : app.WD f x) : β :=
    Classical.choose wd.isInDomain
```

Impossible to build a term using a partial operator without supplying a WD proof.

Encoding B in Lean

WD as metavariables, reified as goals

When BARReL encodes a B formula containing a partial operator:

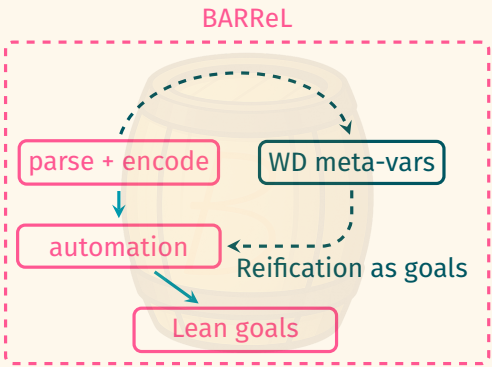


Encoding B in Lean

WD as metavariables, reified as goals

When BARREL encodes a B formula containing a partial operator:

1. a **metavariable** ?wd is inserted at the WD-proof slot.

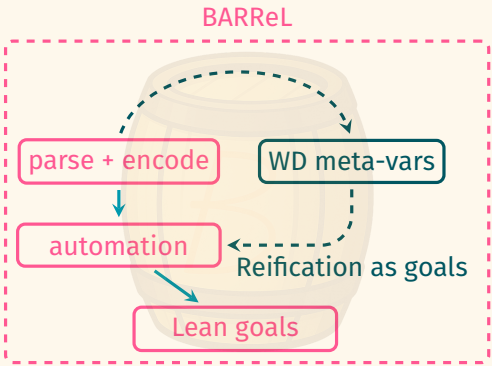


Encoding B in Lean

WD as metavariables, reified as goals

When BARREL encodes a B formula containing a partial operator:

1. a **metavariable** ?wd is inserted at the WD-proof slot.
2. all metavariables are collected and reified as subgoals.

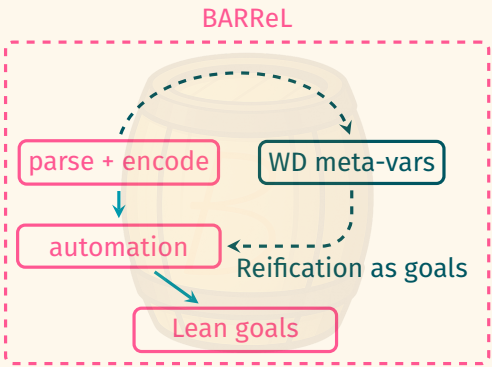


Encoding B in Lean

WD as metavariables, reified as goals

When BARREL encodes a B formula containing a partial operator:

1. a **metavariable** ?wd is inserted at the WD-proof slot.
2. all metavariables are collected and reified as subgoals.
3. an automation pass attempts to discharge routine WD subgoals.

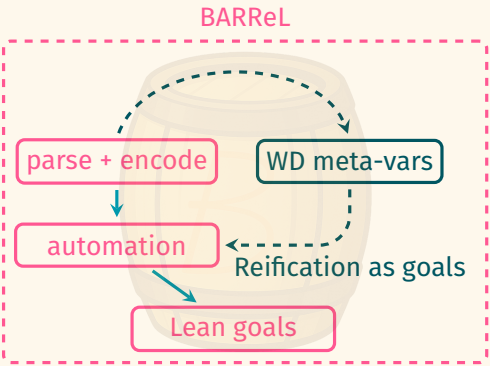


Encoding B in Lean

WD as metavariables, reified as goals

When BARREL encodes a B formula containing a partial operator:

1. a **metavariable** ?wd is inserted at the WD-proof slot.
2. all metavariables are collected and reified as subgoals.
3. an automation pass attempts to discharge routine WD subgoals.
4. The remaining goals are presented to the user.

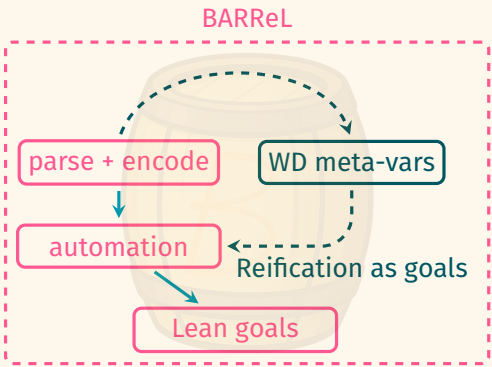


Encoding B in Lean

WD as metavariables, reified as goals

When BARREL encodes a B formula containing a partial operator:

1. a **metavariable** ?wd is inserted at the WD-proof slot.
2. all metavariables are collected and reified as subgoals.
3. an automation pass attempts to discharge routine WD subgoals.
4. The remaining goals are presented to the user.



$\Longrightarrow$ **no PO step can quietly rely on an ill-defined term.**

Encoding B in Lean

Automation

WD tactics and categorization for WD lemmas:

Tactic	WD property
`wd_min`	bounded below + non-empty
`wd_max`	bounded above + non-empty
`wd_app`	functional + in domain
`wd_card`	finite

- Get rid of elementary set-theoretic facts.
- Non-trivial facts are left to the user, with full Mathlib at hand.

Encoding B in Lean

Automation

```
@[wd_min]
theorem min.WD_of_finite [LinearOrder α] {S A : Set α} (h : S ∈ FIN₁ A) :
  min.WD S

@[wd_app]
theorem app.WD_of_overload {A C : Set α} {B D : Set β} {f g : SetRel α β} {x : α}
  (hf : f ∈ A ⇸ B) (hg : g ∈ C ⇸ D) (hx : x ∈ dom f ∨ x ∈ dom g) :
    app.WD (f <+ g) x
```

Encoding B in Lean

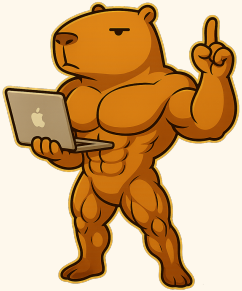
Automation

```
@[wd_min]
theorem min.WD_of_finite [LinearOrder α] {S A : Set α} (h : S ∈ FIN₁ A) :
  min.WD S

@[wd_app]
theorem app.WD_of_overload {A C : Set α} {B D : Set β} {f g : SetRel α β} {x : α}
  (hf : f ∈ A ⇸ B) (hg : g ∈ C ⇸ D) (hx : x ∈ dom f ∨ x ∈ dom g) :
    app.WD (f <+ g) x
```

User-extensible: any new lemma tagged `@[wd_*]` is picked up automatically.

Demo

Live!



Demo time!

Conclusion

What we have, where we're going

- BARReL is lightweight and modular (≈ 1.3*k* LoC impl., ≈ 1.3*k* LoC lib.),
- By-construction WD discipline,
- https://github.com/VTrelat/BARReL

Conclusion

What we have, where we're going

- BARReL is lightweight and modular (≈ 1.3*k* LoC impl., ≈ 1.3*k* LoC lib.),
- By-construction WD discipline,
- https://github.com/VTrelat/BARReL

Future work, ordered by difficulty:

- Cover more of the B language,
- Beef up automation,
- Reduce WD redundancy and improve scalability,
- Implement a **native**, **verified** PO generator.