Rust Toolchain for Event-B

Parser · Pretty printer · Static checker · LSP · CLI

Denis Efremov[1] Ilya Shchepetkov[2]

May 18th, 2026 — 13th Rodin User and Developer Workshop, Tokyo

[1]Ivannikov Institute for System Programming of the RAS · efremov@ispras.ru
[2]Kaspersky Lab · ilya.shchepetkov@17451k.space

Motivation — three real workflow pains

Supporting the development workflow, not the research workflow.

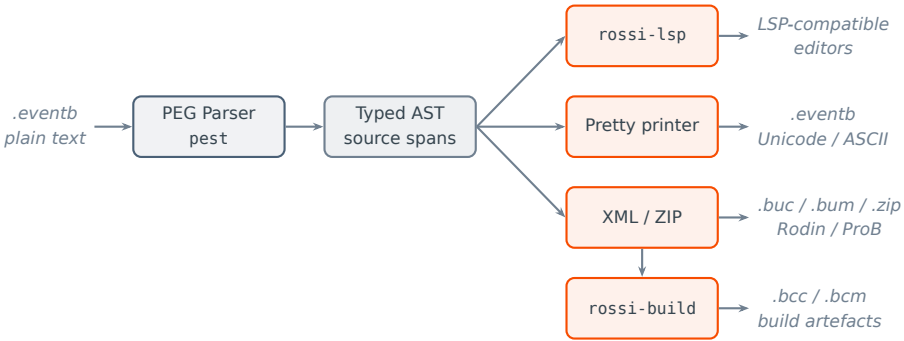
- **Sharing models.** A `.zip` arrives without `.bcm` files — ProB will not open it. There is no quick way to check, from the command line, that a model was built and that all proof obligations were discharged.
- **Teaching.** 30 students × small models × every week. The bottleneck is opening each submission in Rodin to check it, not the proof effort itself.
- **Engineering hygiene.** Text-based version control, code review with diffs, continuous integration — none of these work well on `.zip`-shaped models.

Why we cannot just patch Rodin

- Event-B tooling is provided by the Rodin platform, built on **Eclipse** and the Java SDK
- The Rodin formula parser (HHU STUPS' `rodin-eventb-ast`) is on Maven Central, but covers only *formulas* — not the Camille textual notation
- The static checker and proof-obligation generator are distributed only as Eclipse/OSGi plug-ins inside Rodin
- No language server exists outside Eclipse; CamilleX still depends on Xtext
- Modern development practice relies on text-based version control, continuous integration, and editor-agnostic protocols such as LSP

> We need **text-first tooling** that does not depend on Eclipse.

Architecture overview



- **rossi** (parser · AST · pretty · XML/ZIP)
- **rossi-lsp** (language server)
- **rossi-build** (static checker)
- **rossi-cli** (the `rossi` binary)

Parser and abstract syntax tree

- PEG grammar with `pest`: a single declarative file of ∼620 lines covers the full Event-B mathematical language
- Both Unicode and ASCII notations supported, following the Rodin Keyboard conventions
- **Error recovery** is enabled, so the language server can produce diagnostics on malformed input
- AST uses Rust `enums` and `structs` with source spans on every node; serialisation via `serde` is optional
- Exposed as a Rust library — usable as a dependency from other projects

Category	#	Examples
Binary	33	$+ \; \cup \; \cap \; \mapsto \; \triangleleft$
Unary	6	*dom ran POW* $\sim$
Comparison	10	$= \; \neq \; < \; \leq \; \in \; \subseteq$
Logical	4	$\wedge \; \vee \; \Rightarrow \; \Leftrightarrow$
Quantifier	2	$\forall \; \exists$
Function types	9	$\rightarrow \; \leftrightarrow \; \twoheadrightarrow$
Total	**64**	

For example: $\wedge$ and &, $\Rightarrow$ and =>, $\mapsto$ and |->

```
$ cat lock.eventb
MACHINE lock
SEES
    lock_ctx
VARIABLES
    open
INVARIANTS
    @inv1 open ∈ BOOL
EVENTS
    EVENT INITIALISATION
    THEN
        @act1 open ≔ FALSE
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END
END

$
```

```
$ cat lock.eventb
MACHINE lock
SEES
    lock_ctx
VARIABLES
    open
INVARIANTS
    @inv1 open ∈ BOOL
EVENTS
    EVENT INITIALISATION
    THEN
        @act1 open ≔ FALSE
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END
END

$ rossi print lock.eventb lock_ctx.eventb -o lock.zip
```

```
$ cat lock.eventb
MACHINE lock
SEES
    lock_ctx
VARIABLES
    open
INVARIANTS
    @inv1 open ∈ BOOL
EVENTS
    EVENT INITIALISATION
    THEN
        @act1 open ≔ FALSE
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END
END

$ rossi print lock.eventb lock_ctx.eventb -o lock.zip
```

```
MACHINE lock
SEES
    lock_ctx
VARIABLES
    open
INVARIANTS
    @inv1 open ∈ BOOL
EVENTS
    EVENT INITIALISATION
    THEN
        @act1 open ≔ FALSE
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END
END

$ rossi print lock.eventb lock_ctx.eventb -o lock.zip

$ unzip -l lock.zip
```

```
    EVENT INITIALISATION
    THEN
        @act1 open ≔ FALSE
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END
END

$ rossi print lock.eventb lock_ctx.eventb -o lock.zip

$ unzip -l lock.zip
Archive:  lock.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     1274  05-18-2026 04:35   lock.bum
      478  05-18-2026 04:35   lock_ctx.buc
---------                     -------
     1752                     2 files
```

```
    THEN
        @act1 open := FALSE
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open := TRUE
    END
END

$ rossi print lock.eventb lock_ctx.eventb -o lock.zip

$ unzip -l lock.zip
Archive:  lock.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     1274  05-18-2026 04:35   lock.bum
      478  05-18-2026 04:35   lock_ctx.buc
---------                     -------
     1752                     2 files

$
```

```
    THEN
        @act1 open := FALSE
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open := TRUE
    END
END

$ rossi print lock.eventb lock_ctx.eventb -o lock.zip

$ unzip -l lock.zip
Archive:  lock.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     1274  05-18-2026 04:35   lock.bum
      478  05-18-2026 04:35   lock_ctx.buc
---------                     -------
     1752                     2 files

$ rossi print lock.zip -o ascii/ --ascii
```

```
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open := TRUE
    END
END

$ rossi print lock.eventb lock_ctx.eventb -o lock.zip

$ unzip -l lock.zip
Archive:  lock.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     1274  05-18-2026 04:35   lock.bum
      478  05-18-2026 04:35   lock_ctx.buc
---------                     -------
     1752                     2 files

$ rossi print lock.zip -o ascii/ --ascii

$
```

```
    END

    EVENT unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open := TRUE
    END
END
```

```
$ rossi print lock.eventb lock_ctx.eventb -o lock.zip

$ unzip -l lock.zip
Archive:  lock.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     1274  05-18-2026 04:35   lock.bum
      478  05-18-2026 04:35   lock_ctx.buc
---------                     -------
     1752                     2 files

$ rossi print lock.zip -o ascii/ --ascii

$ diff lock.eventb ascii/lock.eventb
```

```
---------  ---------- -----   ----
     1274  05-18-2026 04:35   lock.bum
      478  05-18-2026 04:35   lock_ctx.buc
---------                     -------
     1752                     2 files

$ rossi print lock.zip -o ascii/ --ascii

$ diff lock.eventb ascii/lock.eventb
7c7
<     @inv1 open ∈ BOOL
---
>     @inv1 open : BOOL
11c11
<         @act1 open ≔ FALSE
---
>         @act1 open := FALSE
18c18
<         @act1 open ≔ TRUE
---
>         @act1 open := TRUE
20a21
>
```

Round-trip with Rodin

parse ∘ *pretty* ∘ *parse* = *parse* — round-trip is **total** on the entire test corpus

Pretty printer

- Two output modes: **Unicode** (accepted by Rodin's Camille parser) and **ASCII** (for tooling that cannot render Unicode)
- Exposed as LSP `textDocument/formatting` — any LSP editor gets "Format Document"
- Left inverse of the parser, modulo source spans

XML interop

- Same AST from `.eventb`, `.buc`, and `.bum` files
- `.zip` project archives accepted as input *and* output
- **Byte-stable XML emitter** → meaningful `git diff`
- Built on `quick-xml`

Typical use: edit as text · version-control · convert to `.zip` for interactive proof.

Static checker (`rossi-build`)

- **Type inference** — variables, parameters, set comprehensions
- **Scoping and well-formedness** checks across contexts and machines
- **Normalisation** — of set-comprehension and overwrite shorthands
- Emits Rodin's `.bcc` and `.bcm` build artefacts and repacks them into the project archive
- Consumed by **Rodin's prover *and* ProB** — inputs unchanged
- Runs in CI — no Eclipse on the build machine

```
$ diff good/M.eventb bad/M.eventb
3c3
<     open
---
>     opn

$
```

```
$ diff good/M.eventb bad/M.eventb
3c3
<     open
---
>     opn

$ rossi validate good.zip
✓ good.zip:M.bum - Valid Machine 'M'
```

```
$ diff good/M.eventb bad/M.eventb
3c3
<     open
---
>     opn

$ rossi validate good.zip
√ good.zip:M.bum - Valid Machine 'M'

$ rossi validate bad.zip
```

```
$ diff good/M.eventb bad/M.eventb
3c3
<     open
---
>     opn

$ rossi validate good.zip
✓ good.zip:M.bum - Valid Machine 'M'

$ rossi validate bad.zip
✓ bad.zip:M.bum - Valid Machine 'M'
! bad.zip (M.opn) - [EB006] could not infer variable type from invariants
✗ bad.zip (M.inv1) - [EB018] unknown identifier 'open' in invariant predicate
```

Language server (`rossi-lsp`)

Cross-file analysis follows the refinement chain — resolution, rename, and find-references traverse `EXTENDS` · `REFINES` · `SEES`

Standard LSP features delivered

diagnostics · completion · hover · signature help · go-to-definition · find-references · workspace symbols · rename · semantic tokens · folding · formatting · code actions (ASCII ↔ Unicode)

Diagnostics survive malformed input

Parser error recovery feeds `textDocument/publishDiagnostics` — one bad line does not stop analysis of the rest of the file.

Works in any LSP client

VS Code · Neovim · Emacs · Helix · …

and over SSH, where Eclipse cannot run.

Workspace model

Buffers stored as ropes; multi-document analysis follows the refinement chain.

In your editor

counter.eventb

```
1  MACHINE counter
2    REFINES counter_abs
3    SEES counter_ctx
4    VARIABLES n
5    INVARIANTS
6      @inv1 n ∈ ℕ
7      @inv2 n ≤ MAX
8    EVENTS
9      INITIALISATION
10       BEGIN n :: ℕ
```

- **Hover & go-to-def** on MAX
  crosses SEES into `counter_ctx`
- **Diagnostic** on the malformed assignment
  the rest of the file still parses (error recovery)
- Same tooling in **VS Code, Neovim, Emacs**
  and over SSH, where Rodin cannot run

Command-line interface

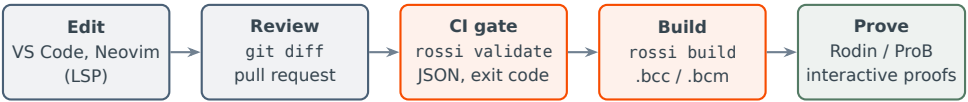
One binary, four verbs.

rossi — terminal

```
$ rossi <verb>
        |-- validate  syntax check · JSON or text · CI exit codes
        |-- print     format & convert · .zip ↔ .eventb · Unicode / ASCII
        |-- build     emit .bcc / .bcm · repacked into the archive
        `-- lsp       launch language server on stdin / stdout
```

- Accepted inputs: `.eventb`, `.buc`, `.bum`, `.zip`
- `validate` powers CI — JSON output and exit codes are designed for it
- `print` and `build` share the same Rust crates the language server uses

Integration into a development workflow



- Models authored as text and reviewed in **pull requests** — the same artefacts feed CI and the prover
- Late-stage tweak? Open in a terminal editor, fix it, repackage, re-run ProB — no full Rodin launch

```
$ cat lock.eventb
```

```
    open
INVARIANTS
    @inv1 open ∈ BOOL
EVENTS
    EVENT INITIALISATION
    THEN
        @act1 open ≔ FALSE
    END

    EVENT do_unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END

    EVENT do_lock
    WHERE
        @grd1 open = TRUE
    THEN
        @act1 open ≔ FALSE
    END
END
```

```
INVARIANTS
    @inv1 open ∈ BOOL
EVENTS
    EVENT INITIALISATION
    THEN
        @act1 open ≔ FALSE
    END

    EVENT do_unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END

    EVENT do_lock
    WHERE
        @grd1 open = TRUE
    THEN
        @act1 open ≔ FALSE
    END
END

$ rossi build lock.zip -o built.zip
```

```
    EVENT INITIALISATION
    THEN
        @act1 open = FALSE
    END

    EVENT do_unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open = TRUE
    END

    EVENT do_lock
    WHERE
        @grd1 open = TRUE
    THEN
        @act1 open = FALSE
    END
END

$ rossi build lock.zip -o built.zip
rossi build: wrote lock.zip -> built.zip (1 file(s), 0 error diagnostic(s))

$ animate built.zip --steps 5 --invariants
```

```
    THEN
        @act1 open ≔ FALSE
    END

    EVENT do_unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open ≔ TRUE
    END

    EVENT do_lock
    WHERE
        @grd1 open = TRUE
    THEN
        @act1 open ≔ FALSE
    END
END

$ rossi build lock.zip -o built.zip
rossi build: wrote lock.zip -> built.zip (1 file(s), 0 error diagnostic(s))

$ animate built.zip --steps 5 --invariants
```

```
    EVENT do_unlock
    WHERE
        @grd1 open = FALSE
    THEN
        @act1 open := TRUE
    END

    EVENT do_lock
    WHERE
        @grd1 open = TRUE
    THEN
        @act1 open := FALSE
    END
END

$ rossi build lock.zip -o built.zip
rossi build: wrote lock.zip -> built.zip (1 file(s), 0 error diagnostic(s))

$ animate built.zip --steps 5 --invariants
08:36:13.026 [main] INFO de.prob.cli.Installer -- Installing CLI binaries to a new temporary
 directory
08:36:13.392 [main] INFO animate.Animate -- Load Event-B Machine
Machine: lock
```

```
Prolog: SICStus 4.8.0 (arm64-darwin-20.1.0): Sun Dec  4 14:17:07 CET 2022

Animation steps:
do_unlock()
do_lock()
do_unlock()
do_lock()
do_unlock()

Current state:
( open=TRUE )

Coverage properties:
         - 'deadlocked:0'
         - 'invariant_violated:0'
         - 'invariant_not_checked:0'
         - 'open:0'
         - 'live:3'
         - 'total:3'
Covered operations:
         - 'INITIALISATION:1'
         - 'do_unlock:1'
         - 'do_lock:1'
```

Testing strategy

Property-based testing

- Random AST generators for `Expression`, `Predicate`, `Action`, `Event`, `Context`, `Machine`
- 500 cases per expression / predicate, 200 per context / machine, in both Unicode and ASCII
- Property: $parse \circ pretty \circ parse = parse$, modulo source spans
- Covers all 33 binary ops, 6 unary ops, lambdas, set comprehensions, quantifiers, refinement clauses

Corpus validation

- **68 public Rodin projects** — ARINC 653 (avionics OS), ERTMS / ETCS (rail signalling), HIMACF (access control), ABZ case studies
- Three independent oracles:
  1. Semantic diff of `rossi-build` output vs. Rodin's bundled `.bcc` / `.bcm`
  2. Replay through **ProB**
  3. Agreement with **Rodin implementation**

Conclusion and future work

Available now

- Parser and round-trip pretty printer
- Interoperability with Rodin `.zip` archives
- Static checker `rossi-build`
- Language server and command-line tools
- VS Code extension

Planned work

- Semantic validation, type checking, well-definedness
- Generation of proof obligations
- Plugin interface for third-party analyses
- LLM-based proof automation

  keep proofs valid across cosmetic changes — variable renames, Rodin version bumps — without manual replay

`github.com/eventb-rossi/rossi` • MIT / Apache-2.0