Towards the formally validated crystallographic software

S. Gražulis A. Merkys A. Vaitkus K. Petrauskas L. Laibinis

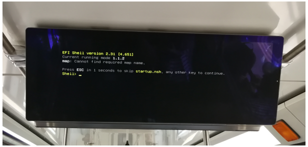
Melbourne, 2023

Vilnius University
Institute of Biotechnology, Life Sciences Center
Institute of Informatics, Faculty of Mathematics and Computer Science

Id: slides.tex 2252 2023-08-26 00:47:37Z saulius August 26, 2023

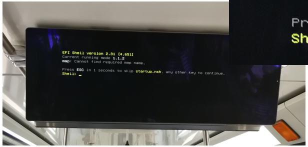
Software is ubiquitous...

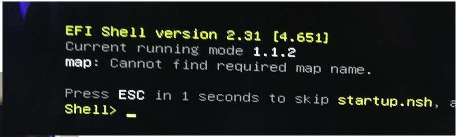
... and so, it seems, are software bugs ...



Software is ubiquitous...

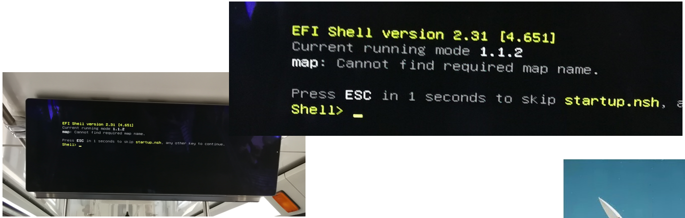
... and so, it seems, are software bugs ...



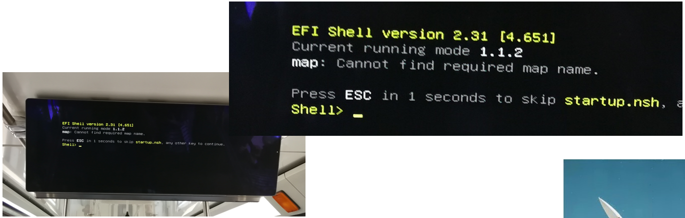


Software is ubiquitous...

... and so, it seems, are software bugs ...



Accumulating 0.1 s intervals for 100 h (in 24 bit binary) resulted in missing the target by 0.3 s...



*"/The/ program, which was not part of a conventional data processing package, converted the anomalous pairs (**I+** and I–) to (**F–** and F+), thereby introducing a sign change. /As a result/ the structures reported had the wrong hand."*

Accumulating 0.1 s intervals for 100 h (in 24 bit binary) resulted in missing the target by 0.3 s...

Understandable software

We want our source code to be readable! Not like this:

```
$s=2;
$d=500;
$w="A";$_='ZIsHPX=$s-Z*Z;$|C;J"sH=\nZ.";O!XNJ"0"x$d,"\n";exit}QZNpush
(F,Z%10PZIZD)}QXNpush(@W,X%10PXIXD)}subT{GMw>MW)OMw!=MWPZ=Mw;QE1NGZV>B)
OZV!=BPZK}1}subY{my(FPZ=0;X=Mw+1;QX>ZNXV+=ZV*S;X[E1]IXVDPXV%C0;E+}MYKO!X
[MY]PF}Q$dKNLF;S=2;@T=Y;@W=(0,0,@WPSC;QSNAOTNF=(KS,FPlast}S++}AZ[0]K;Z=0;S
=MW+1;QZ-SNB+=9-ZV;OB>CONB-C0;Z[E1]K}E+}Q!U[MW]NMWK};JX[0]}J"\n";
';foreach$s(qw/ L(S,@TPLY; UV =1*.1 Z+ @Y return( qrt($s) =R(
prR -- @w= $# )
{ if( ); Te( int Ul Wl Xi [Z] Yi Zh wh $w
/){s;$w;$s;g;$w++}eval;
```

Understandable software

We want our source code to be readable! Not like this:

```
$s=2;
$d=500;
$w="A";$_='ZIsHPX=$s-Z*Z;$|C;J"sH=\nZ.";O!XNJ"0"x$d,"\n";exit}QZNpush
(F,Z%10PZIZD)}QXNpush(@W,X%10PXIXD)}subT{GMw>MW)OMw!=MWPZ=Mw;QE1NGZV>B)
OZV!=BPZK}1}subY{my(FPZ=0;X=Mw+1;QX>ZNXV+=ZV*S;X[E1]IXVDPXV%C0;E+}MYKO!X
[MY]PF}Q$dKNLF;S=2;@T=Y;@W=(0,0,@WPSC;QSNAOTNF=(KS,FPlast}S++}AZ[0]K;Z=0;S
=MW+1;QZ-SNB+=9-ZV;OB>CONB-CO;Z[E1]K}E+}Q!U[MW]NMWK};JX[0]}J"\n";
';foreach$s(qw/ L(S,@TPLY; UV =1*.1 Z+ @Y return( qrt($s) =R(
prR -- @w= $# )
{ if( ); Te( int Ul Wl Xi [Z] Yi Zh wh $w
/){s;$w;$s;g;$w++}eval;
```

Daniel Rinehart, a self-uncompressing square root finder and custom bignum library.

http://www.foo.be/docs/tpj/issues/vol2_3/tpj0203-0012.html

Formal specifications

Formal methods:

1. Allow to *specify* software behaviour formally;
2. Allow to *prove* that software conforms to specification;
3. Allow to *run* the software with the proven properties;

Formal specifications

Formal methods:

1. Allow to *specify* software behaviour formally;
2. Allow to *prove* that software conforms to specification;
3. Allow to *run* the software with the proven properties;

```
function Build_Group (E : Ring_Element) return Group
  with
  Post => Is_Group (Build_Group'Result);
```

Formal specifications

Formal methods:

1. Allow to *specify* software behaviour formally;
2. Allow to *prove* that software conforms to specification;
3. Allow to *run* the software with the proven properties;

```
function Is_Group (G : Group) return Boolean
is (Has_Identity (G) and then
      All_Elements_Have_Inverses (G) and then
      Is_Closed_On_Multiplication (G)
   )
```

Formal specifications

Formal methods:

1. Allow to *specify* software behaviour formally;
2. Allow to *prove* that software conforms to specification;
3. Allow to *run* the software with the proven properties;

```
function Is_Group (G : Group) return Boolean
is (Has_Identity (G) and then
        All_Elements_Have_Inverses (G) and then
        Is_Closed_On_Multiplication (G)
   )

function Is_Closed_On_Multiplication (G : Group) return Boolean
is (for all E of G =>
        (for all F of G => (Belongs_To (E*F, G))))
```

Formal specifications

Formal methods:

1. Allow to *specify* software behaviour formally;
2. Allow to *prove* that software conforms to specification;
3. Allow to *run* the software with the proven properties;

```
function Is_Group (G : Group) return Boolean
is (Has_Identity (G) and then
      All_Elements_Have_Inverses (G) and then
      Is_Closed_On_Multiplication (G)
   )

function Belongs_To (E : Element; G : Group) return Boolean
is (for some F of G => (E = F))
```

Formal systems for software development

A non-exhaustive list of tools:

1. Proof assistants
   - Isabelle/HOL;
   - Coq/Gallina;
2. Software development systems (proovers)
   - Ada/SPARK
   - C#/Spec#;
   - C/Frama-C;
   - Daphny/Boogie;
   - Java/KeY;
   - Java/JML;
   - Java/EST;
   - Java/Sooth+ByteBack+Boogie

Formal systems for software development

A non-exhaustive list of tools:

1. Proof assistants
    - Isabelle/HOL;
    - Coq/Gallina;
2. Software development systems (proovers)
    - **Ada/SPARK**
    - C#/Spec#;
    - C/Frama-C;
    - Daphny/Boogie;
    - Java/KeY;
    - Java/JML;
    - Java/EST;
    - Java/Sooth+ByteBack+Boogie

Why Ada/SPARK?

1. Durable design – first designed in 1983!
2. Modern language – latest standard is Ada 2022;
3. Mostly backwards compatible;
4. Good F/LOSS compiler available – GNAT;
5. Ada is statically very strictly typed;
6. Programs are easy to read (Level (Ada) > Level (C));
7. Ada & SPARK has a rich type system;
8. Language level concurrent programming;
9. Produces fast optimised native code, links with any language;
10. SPARK subset takes computer arithmetic into account;
11. Not controlled by any private company;

Why is Ada not popular (yet)?

1. The language is complex and difficult to implement;
2. No good compilers in the 1990's;
3. Procured by the DOD, used for "war fighting software";
4. Poor academic outreach in the 20th century;

Why is Ada not popular (yet)?

1. ~~The language is complex and difficult to implement~~;
2. The language is rich and convenient to program/design in;
3. ~~No good compilers in the 2020's;~~
4. Very nice compiler and dev. system available: `gnat`;
5. Procured by the DOD, used for "war fighting software";
6. Used for mission-critical software (avionics, spacecraft ctrl., railways, plant ctrl...)
7. Poor academic outreach in the 21st century;
8. SPARK allows formal verification of the code (!);

Why is Ada not popular (yet)?

1. ~~The language is complex and difficult to implement~~;
2. The language is rich and convenient to program/design in;
3. ~~No good compilers in the 2020's;~~
4. Very nice compiler and dev. system available: `gnat`;
5. Procured by the DOD, used for "war fighting software";
6. Used for mission-critical software (avionics, spacecraft ctrl., railways, plant ctrl...)
7. <u>Poor academic outreach in the 21st century</u>;
8. SPARK allows formal verification of the code (!);

The algorithm to work with

The group reconstruction algorithm: generate, when presented with a subset of elements from some existing finite group G, a (smallest) subgroup $H \leq G$ containing those elements:

$$\{g_1, ..., g_n\}, \forall g_i : g_i \in G \to H \leq G : \text{Is\_Group}(H) \wedge \forall g_i : g_i \in H$$

Uses of this algorithm:

- check symmetry operators of a CIF file (for the COD);
- determine symmetry of special position;
- check whether an atom is on a sp. pos.;
- constrain an atom to a sp. pos. for refinement (PD?);
- analyse disorder around a special position;

(Grosse-Kunstleve 1999)

Formal proofs of the the algorithms in use

Require: H – a subgroup of a finite group G
Require: g – an element of the finite group G, $g \in G$
Ensure: The list L of the operators of a subgroup $L \leq G$ without duplicates
Ensure: L contains both g and the elements of H

```
procedure SimpleBuilder(H, g)
    ▷ Build a space group generated by H and g
    L ← [e, h_1, h_2, ..., h_n], where ∀i. h_i ∈ H
    L_new ← [g]
    while L_new is not empty do
        g' ← head(L_new)
        L_new ← tail(L_new)
        L ← append(L, g')
        for all h' ∈ L do
            g'' ← h' ⊗ g'
            if g'' ∉ L ∪ L_new then
                L_new ← append(L_new, g'')
            end if
        end for
    end while
    return L
end procedure
```

Figure 2
The optimized simple space-group-builder (core) algorithm.

```
have "subgroup R G"
proof -
    have R_subset: "R ⊆ carrier G" sorry
    moreover have R_m_closed: "⋀x y. ⟦x ∈ R; y ∈ R⟧ ⟹ x ⊗ y ∈ R" sorry
    moreover have R_one_closed: "1 ∈ R" sorry
    moreover have R_m_inv_closed: "⋀x. x ∈ R ⟹ inv x ∈ R" sorry
    ultimately show "subgroup R G" by (simp add: subgroup_def)
qed
```

```
for my $group_symop (@{$self->{symops}}) {
    do {
        my $product =
            snap_to_crystallographic(
                symop_modulo_1(
                    symop_mul( $group_symop, $test_symop )
                )
            );
        my $product_key = string_from_symop( $product );
        if( !exists $self->{symop_hash}{$product_key} ) {
```

(Petrauskas et al. 2022)

Group theory in Ada/SPARK

examples/group_theory.ads

```
pragma Spark_Mode (On);

generic
   type Element is private;
   Identity : Element;
   with function "*" (E, F: Element) return Element is <>;
```

```
   function Is_Closed_On_Multiplication (G : Group) return Boolean
   is (for all E of G =>
         (for all F of G => (Belongs_To (E*F, G))))
     with Ghost;

   function All_Elements_Have_Inverses (G : Group) return Boolean
   is (for all E of G => Has_Inverse (E, G))
     with Ghost;

   function Is_Group (G : Group) return Boolean
   is (Has_Identity (G) and then
         All_Elements_Have_Inverses (G) and then
         Is_Closed_On_Multiplication (G)
      )
     with Ghost;
```

examples/make_group.ads

```
8   type Ring_Element is mod 37;
```

```
29  function Build_Group (E : Ring_Element) return Group
30    with
31    Post => Is_Group (Build_Group'Result);
```

gnatprove -P main.gpr --report=all make_group.adb

```
make_group.ads:23:14: info: postcondition proved
make_group.ads:27:14: info: postcondition proved
make_group.ads:31:14: info: postcondition proved
group_theory.ads:16:15: info: postcondition proved, in instantiation at make_group.ads:16
```

```
saulius@tasmanijos-velnias spacegroups/ $ ./run_make_group 8
(1, 8, 27, 31, 26, 23, 36, 29, 10, 6, 11, 14)

saulius@tasmanijos-velnias spacegroups/ $ ./run_make_group 7
(1, 7, 12, 10, 33, 9, 26, 34, 16)
```

Current assumptions

... need to be made

```
function Build_Group (G : Group; E : Ring_Element) return Group
```

```
for I in N'First .. NN loop
   declare
      H : Ring_Element := N (I) * T;
   begin
      if not Contains (N (N'First..NN), H) then
         Add_Element (N, NN, H); -- Add the element to the growing group
         Add_Element (L, NL, H); -- Add the element to the candidate list
      end if;
   end;
end loop;
```

Current assumptions

... need to be made

```
function Build_Group (G : Group; E : Ring_Element) return Group
```

```
for I in N'First .. NN loop
   declare
      H : Ring_Element := N (I) * T;
   begin
      if not Contains (N (N'First..NN), H) then
         Add_Element (N, NN, H); -- Add the element to the growing group
         Add_Element (L, NL, H); -- Add the element to the candidate list
      end if;
   end;
end loop;
```

```
   pragma Assume (All_Elements_Have_Inverses (Group (N (N'First .. NN))));
   pragma Assume (Is_Closed_On_Multiplication (Group (N (N'First .. NN))));

   return Group (N (N'First .. NN));
end Build_Group;
```

Conclusions

- Ada/SPARK provide production-ready F/LOSS dev. environment;
- Software functions (Pre/Post) can be formally specified in SPARK;
- Certain properties can be proved automatically; others – with explicit assumptions;
- More properties will be possible to prove in the future;
- Working (library) code can be generated from the verified source;

Vision for the future

- A reusable F/LOSS library of *verified* crystallographic algorithms;
- Stable and future-proof;
- Compatible with any languages and platforms (Ada, C(++), Go, Julia, Rust, Perl, Python, WebAssembly, etc.);
- Make software *readable* and *understandable*;
- Make software a part of documentation for *scientifc inferences* along with human readable texts (papers, presentations, etc.) and databases (COD, etc.)

Acknowledgements

VU LSC IBT (KICIS)

Andrius Merkys[1]
Antanas Vaitkus[1]
Algirdas Grybauskas

VU MIF II (FMG)

Linas Laibinis[1]
Karolis Petrauskas[1]
Irus Grinis
Haroldas Giedra

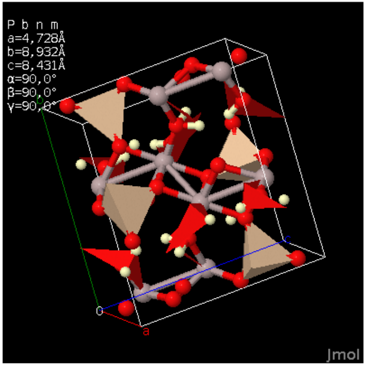
COD Advisory board

Daniel Chateigner
Robert T. Downs
Werner Kaminsky
Armel Le Bail
Luca Lutterotti
Peter Moeck
Peter Murray-Rust
Miguel Quirós

Funding:
Lithuanian-French Program “Gilibert”; CECAM; RCoL grants S-MIP-20-21, S-MIP-23-87, VU Intramural funding.

[1] Co-authors of this work

Thank you!



Coordinates 2207377.cif
Original IUCr paper HTML

http://en.wikipedia.org/wiki/Topaz

http://www.crystallography.net/2207377.html

References I

Gražulis, Saulius et al. (2009). “Crystallography Open Database – an open-access collection of crystal structures”. In: *Journal of Applied Crystallography* 42, pp. 726–729. DOI: 10.1107/S0021889809016690. URL: http://dx.doi.org/10.1107/S0021889809016690.

Grosse-Kunstleve, R. W. (1999). “Algorithms for deriving crystallographic space-group information.”. In: *Acta crystallographica. Section A, Foundations of crystallography* 55, pp. 383–395. DOI: 10.1107/S0108767398010186. URL: http://scripts.iucr.org/cgi-bin/paper?S0108767398010186.

Grosse-Kunstleve, R. W. et al. (1997). “Powder Diffraction Data and Crystal Chemical Information Combined in an Automated Structure Determination Procedure for Zeolites”. In: *J. Appl. Cryst.* 30, pp. 985–995.

Petrauskas, Karolis et al. (May 2022). “Proving the correctness of the algorithm for building a crystallographic space group”. In: *Journal of Applied Crystallography* 55.3, pp. 515–525. DOI: 10.1107/s1600576722003107.