Supporting Information for

Making the InChI FAIR and sustainable while moving to Inorganics

Gerd Blanke,*[a] Jan Brammer,[b] Djordje Baljozovic,[b] Nauman Ullah Khan,[b] Frank Lange,[b] Felix Bänsch,[c] Clare Tovee,[d] Ulrich Schatzschneider,[e] Richard M. Hartshorn,[f] Sonja Herres-Pawlis*[b]

Appendix A. Additional information on programming details

For the first time in three years, a new release of the InChI software (with version number 1.07) was published in December 2023 on a newly created *GitHub* repository.[1] All existing code fragments left by the late developer Igor Pletnev had been integrated into a new code project which served as a foundation for InChI 1.07 development. Since then, more than three thousand bugs, warnings and hints were detected and fixed in the inherited code.

The InChI version 1.07 introduces several new features that can be accessed in test/engineering mode such as:

- support for six additional types of tautomeric transformations
- support for processing of molecules with up to 32766 atoms

Particular emphasis has been given to fixing security issues and bugs, the vast majority of which had been inherited from earlier InChI versions, predominantly 1.06. Moreover, all InChI software versions have been closely monitored by the *Google® oss-fuzz project* team,[2] who have until now detected more than 60 bugs and security issues using various *fuzz testing* techniques.[3]
The source code has been methodically debugged using 64- and 32-bit versions of all available C/C++ compilers in *Microsoft® Windows* and *Ubuntu Desktop* computer operating systems. The following table (Table A1) summarizes all C/C++ compilers which were used on *Microsoft® Windows* and *Ubuntu Linux* platforms for testing and the InChI code development.

C/C++ compiler name	Compiler versions	64-bit	32-bit	Microsoft® Windows	Ubuntu Linux
Microsoft® Visual Studio C++	17.9.1 - 17.10.4	✓	✓	✓	-
GNU GCC	7.5 - 14.1	✓	✓	✓	✓
LLVM/Clang	14.0.3 - 18.0.4	✓	✓	✓	✓

Table A1. Summary of compiler versions used for testing and developing of InChI

The following list shows a summary of all bug and security fixes:

- ☐ 9 functions were rewritten to address issues with function arguments related to arrays of various dimensions
- ☐ 25 blocks of code were rewritten to address memory leaks, buffer overruns, various security issues, and improperly written conditional statements or bitwise operations
- ☐ 33 heap overflow issues were fixed due to use of large array dimensions
- ☐ 157 security bugs related to improper pointer dereferencing were fixed, which might cause crashes or undesired exits
- ☐ 71 memory leaks were fixed
- ☐ 530 places have been marked where optional bounds-checking functions could be used
- ☐ 2480 various bugs and other issues were addressed, such as type conversions and mismatches, removing redundant variables and/or code as well as addressing LLVM/Clang warnings
- ☐ 58 *Google® oss-fuzz* [2] issues were fixed, and 5 additional *Google® oss-fuzz* issues have been partially fixed

Finally, in order to provide a further layer of security, optional support for bounds-checking functions[4] has been provided (in accordance with *Annex K* of the C11 standard[5]).

Because of the primary focus to fix bugs and security fixes, regression tests were run to compare the results between version 1.06 and 1.07 against all PubChem "Compounds" [6], "3D Compounds" [7], and "Substances" [8]. In terms of the nearly 300 million PubChem "Substances" only 2 failed. In terms of invariance tests 2131 substances failed ($6{,}4*10^{-5}$%). Most of the failures were caused by insufficient determination of H tautomerism with the structures.

All binaries/API, instructions for compiling InChI software from the source code, a comprehensive list of known issues as well as many other important technical details and the test suite as Docker container can be found on the landing page of the new InChI *GitHub* repository.[1]

Appendix B Interactive InChI user interfaces

InChI's Graphical User Interface for Windows – WInChI

An interactive tool for Microsoft Windows called *WInChI* has been part of previous releases. It calculates InChIs representations from both MDL Molfiles containing a single chemical structure or SDFiles containing multiple structures.
After the calculation, the chemical structures are displayed together with the canonical InChI atom numbers (marked in yellow in Figure S1). Tautomeric areas within the molecules are indicated with dashed lines which depict the result of the InChI canonicalization. InChI, AuxInfo and InChIKey are displayed below the canvas with helpful information on each of the layers. WInChI calculates the standard InChI by default, but the InChI parameters can be changed in the option menu (Figure S2).

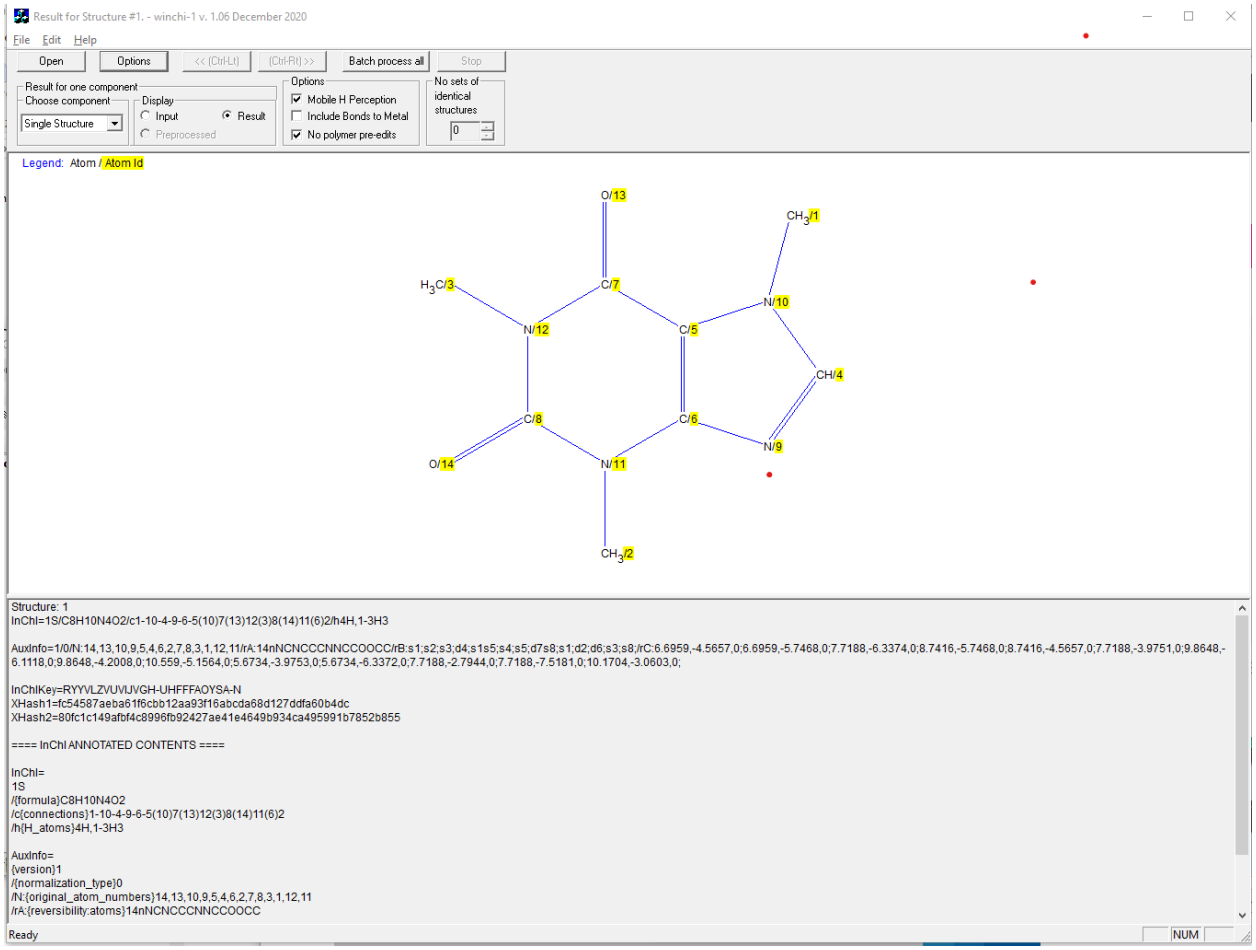


Figure S1: Screenshot of the WInChI canvas

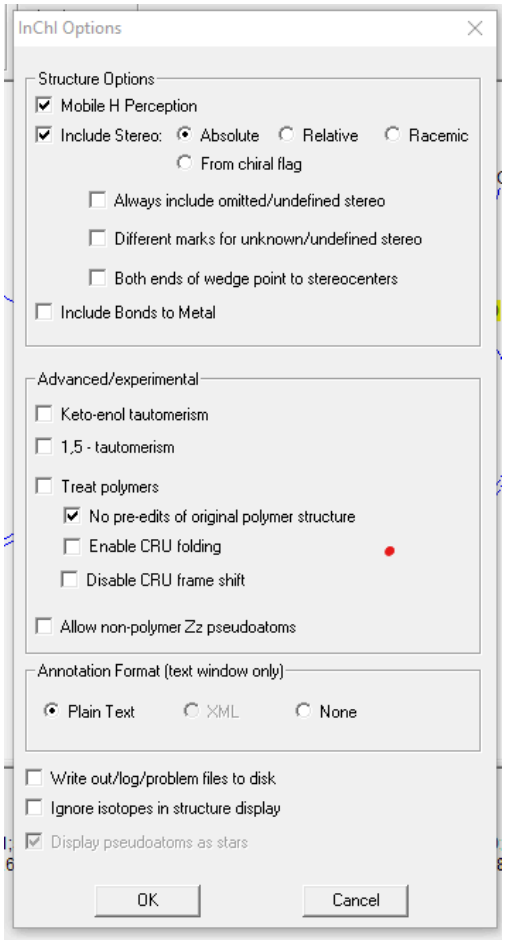


Figure S2: InChI parameter selection tool in WInChI

InChI web demo

This section describes the InChI web demo[9], a new interactive and browser-based application that enables the generation of InChIs and reaction InChIs (RInChIs). Such a browser-native application has the key advantage that there is no need to install any additional software on the user's computer, such as the InChI software suite, WInChI or chemical structure editors that include the InChI library. In addition, the InChI web demo is designed such that all data remains the local browser, thus no web backend infrastructure is required.

The main use case of the application is the generation of InChIs from drawn chemical structures. Structure drawing is facilitated by the powerful Ketcher chemical structure editor[10] and InChI, AuxInfo, InChIKey, and the log of the calculation (for error messages) are shown below the drawing canvas (Figure S3). All InChI parameters and the InChI version (1.07 as well as 1.06 as legacy version) can be selected from the right part of the panel.

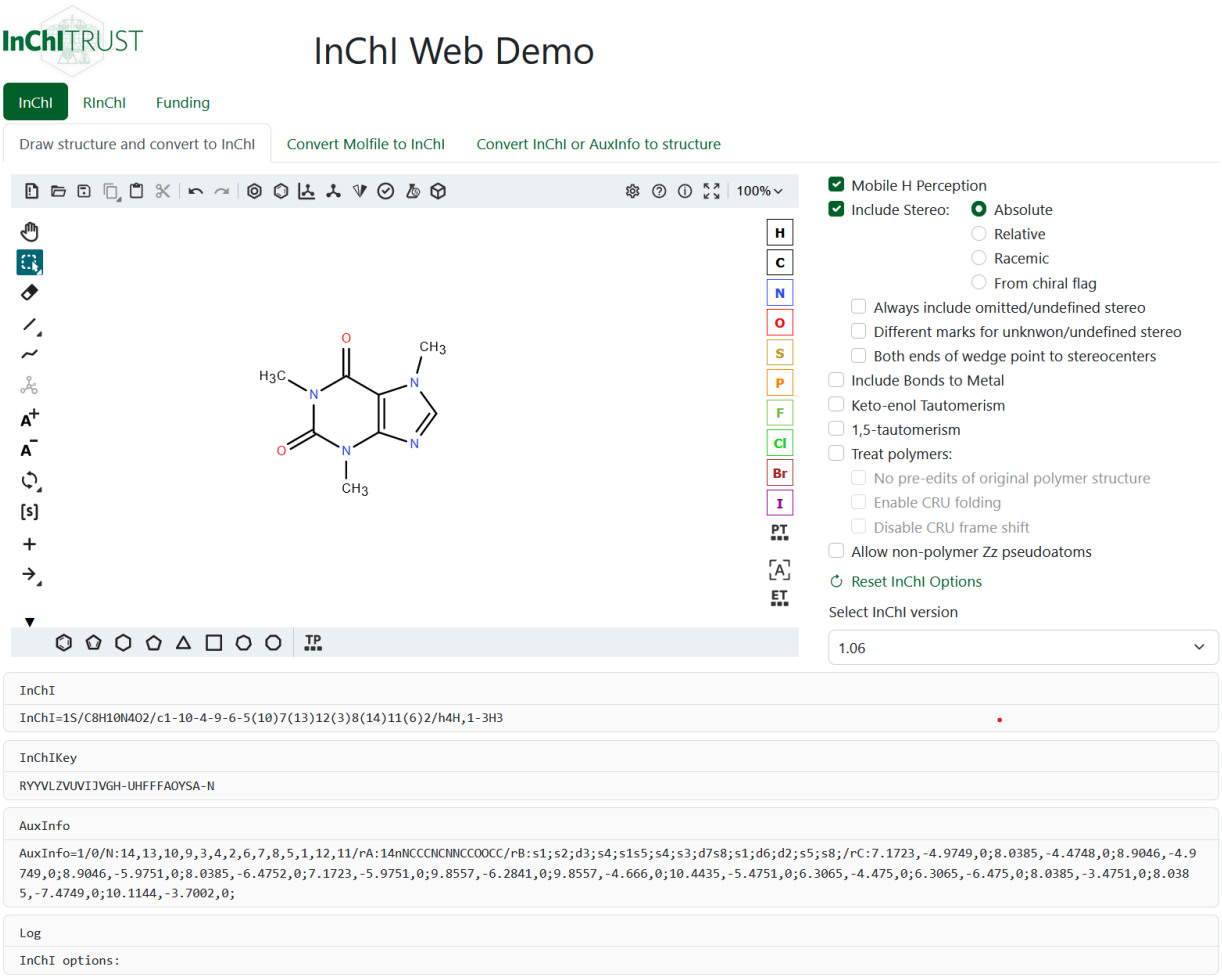


Figure S3: Screenshot of the InChI web demo demonstrating the drawing of a chemical structure, the immediate generation of InChI representation, InChIKey and AuxInfo as well as the interactive selection of the InChI parameters and version.

As an additional functionality, Mofiles can be generated from an InChI representation or AuxInfo, and the evaluation of chemical reactions to obtain RInChIs and the related Long-, Short-, and Web-RInChIKeys beside the reaction AuxInfo (RAuxInfo), which is depicted in Figure S4. Similar to the functionalities related to InChI and Molfiles, a reverse-generation of RXN/RD files from RInChI representations or RAuxInfo was implemented.

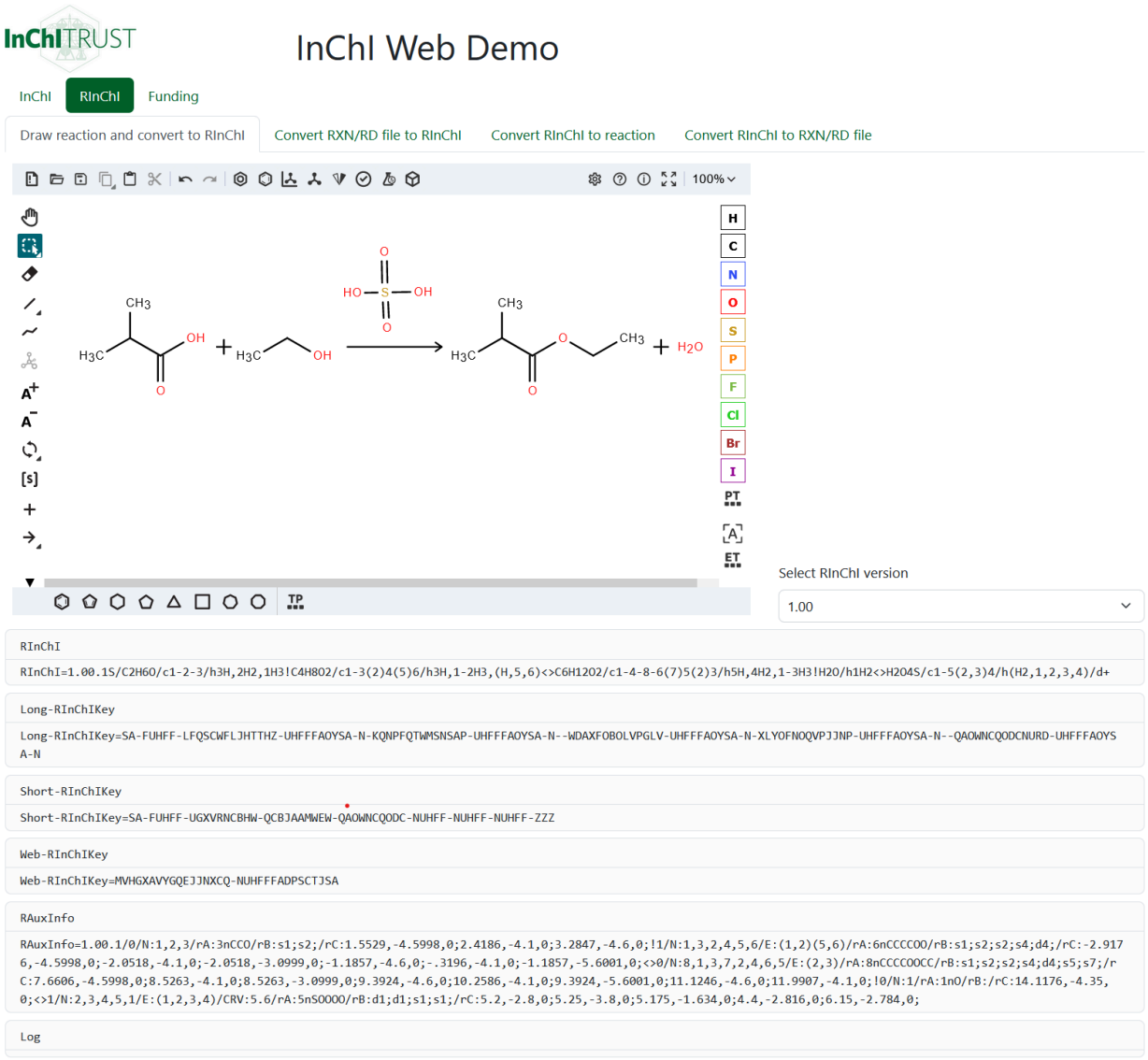


Figure S4: Screenshot of the RInChI view in the InChI web demo. A chemical reaction drawn in the Ketcher structure editor is transformed into a RInChI string, its respective RInChIKeys and RAuxInfo.

On the technical level, the InChI web demo utilizes the WebAssembly technology stack[11] (Wasm) by compiling InChI's C and RInChI's C++ source code to the Wasm binary instruction format using the Emscripten compiler[12]. The design and feasibility of such an application has been demonstrated previously[13,14] and the InChI algorithm has already been integrated in the web versions of major open-source cheminformatics frameworks such as RDKit.js[15] and Indigo[16] *via* the Emscripten tool chain. We would like to emphasize that the port to Wasm opens the InChI algorithm to be used on *any* processor architecture and operating system with a Wasm-compatible web browser.

The InChI web demo will follow the ongoing InChI and RInChI development. Future extensions will introduce a mode to inspect InChI's canonical atom numbering and the labeling of

tautomeric systems similar to WInChI's capabilities, bond types specific to organometallic compounds and atom mappings in reactions for RInChIs. For easier use, specific tutorials such as a RInChI tutorial are being developed.

Appendix C Testing

An important part in our effort of more open and sustainable development of the InChI is to provide a test suite, including test code, data, and documentation. Accordingly, objective and transparent quality criteria were defined that enable convenient and replicable testing. Such a test environment is indispensable for collaborative development, especially with external contributors. Our test suite consists of two kinds of tests.

a) Regression tests

Regression tests are used to detect problems with the *stability* of the InChI across versions. A regression means that the current behavior of a software package deviates from the established proper behavior of a previous version. While developing a feature or fixing a bug, developers repeatedly run these tests to increase their confidence that no regressions are introduced. In Figure S5, the first and second run represent tests that are conducted after alterations to the codebase. During the first run, the output matches the reference state while the one results in a regression, since the output no longer matches the reference. We use the output of the last stable version as a reference for our regression tests.

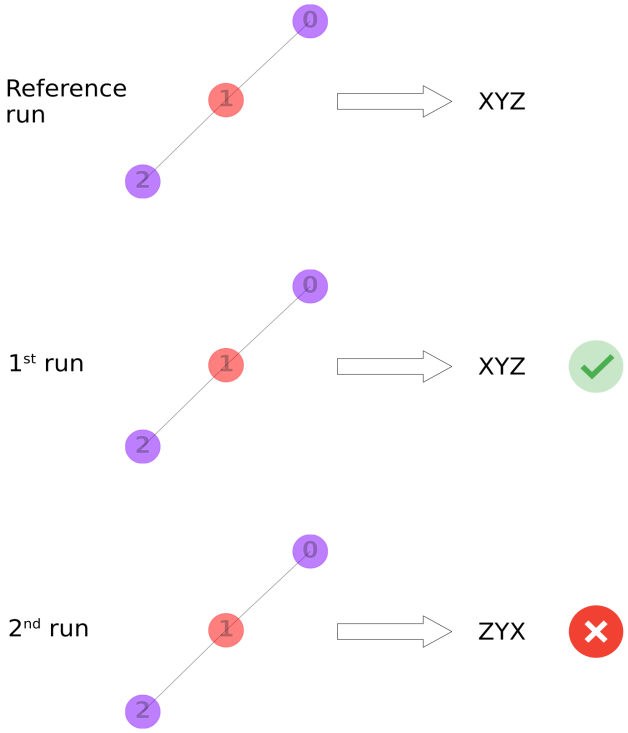


Figure S5: Schematic representation of regression tests: the first test run is successful because the output strings are identical, the second one fails because of a different output

b) Invariance tests

Invariance tests are used to detect problems with the InChI canonicalization algorithm. As such, they increase confidence in an essential feature of the InChI, the unique and stable identification of structures. During an invariance test, the atom indices of a structure are permuted repeatedly and each permutation is expected to result in the same InChI output. In Figure S6, the output of the third permutation deviates from the previous two permutations, resulting in a failed invariance test.

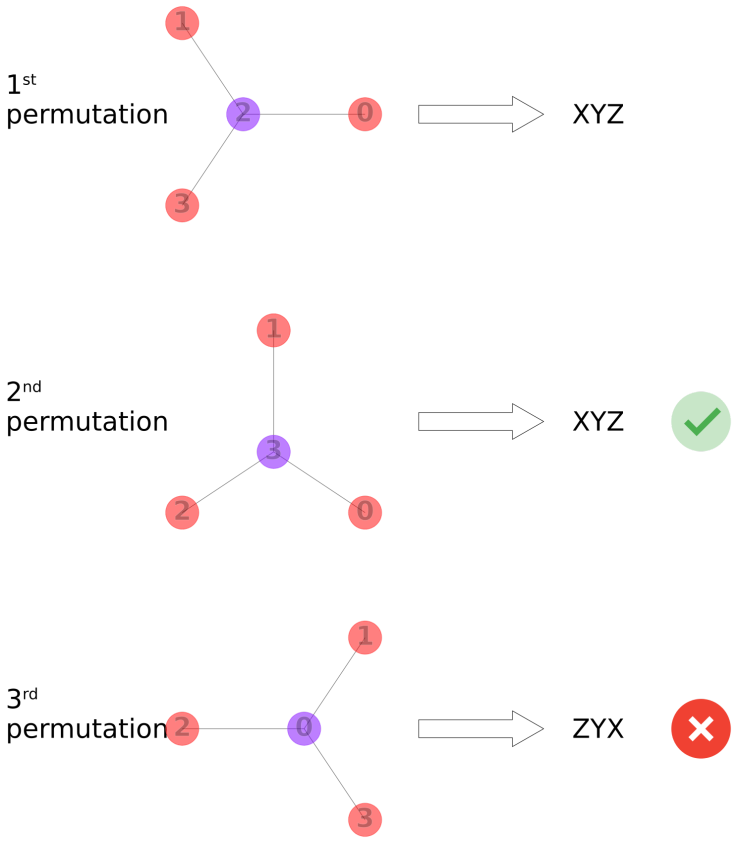


Figure S6: Permutation tests: While the input molecule features a different atom numbering in each test, the test is successful in case the output string remains identical due to a proper canonicalization algorithm. It fails in cases where one output differs from the other permutations.

We run our test suite against two data sets, each serving a unique purpose.

Automated tests

Every contribution to our GitHub repository, by team members or external collaborators, triggers a run of the test suite. The practice of automatically running tests against each contribution is part of the continuous integration (CI).

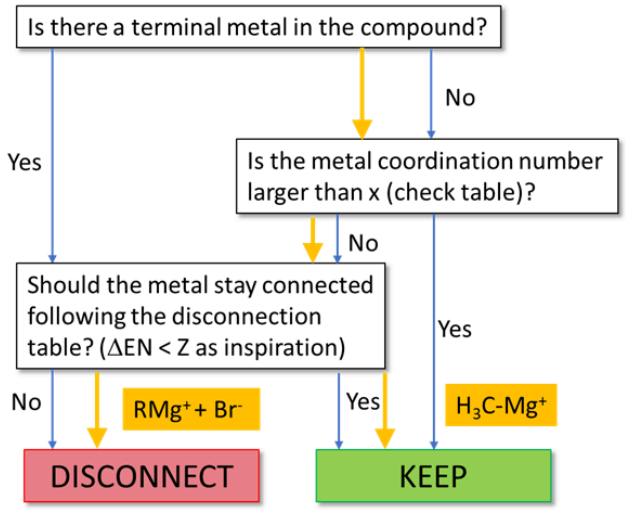
To facilitate quick feedback, automated testing needs to be fast but thorough, to increase confidence (note that those two requirements are antagonistic). We ensure speed by limiting the size of our CI data, resulting in a test suite that runs in a matter of seconds. Furthermore, we make the data informative by including known problematic structures, which have triggered bugs in the past, as well as structures that challenge the InChI algorithmically. The CI data is

included in our GitHub repository to facilitate transparency and re-use. For example, developers can run the test suite against the CI data on their local machines, before contributing on GitHub.

Manual release tests

Prior to each release, the test suite is run manually against the PubChem database. Specifically, tests are run against all PubChem "Compounds"[6], "3D Compounds"[7], and "Substances".[8] The PubChem data are used for testing as is, no pre-processing is applied. These comprehensive test runs are limited to releases, since they require significantly more time than the automated tests. The test results for the current stable release (1.07.0 at the time of writing) can be found here: https://github.com/JanCBrammer/presentation_inchi_tests/blob/main/slides.pdf. Finally, users can run our test suite against their own datasets to pilot new InChI versions before integrating them in their workflows or production environments.

Appendix D More inorganic examples of bond handling



Example: $Br\text{-}Mg\text{-}CH_3$

-> No terminal metal
-> Coordination number = 2
ITERATE over each bond to the metal:
-> Disconnection table says: break Mg-Br and keep $Mg\text{-}CH_3$

-> $H_3C\text{-}Mg^+$ and Br^-

Figure S7: Bond handling for a Grignard compound

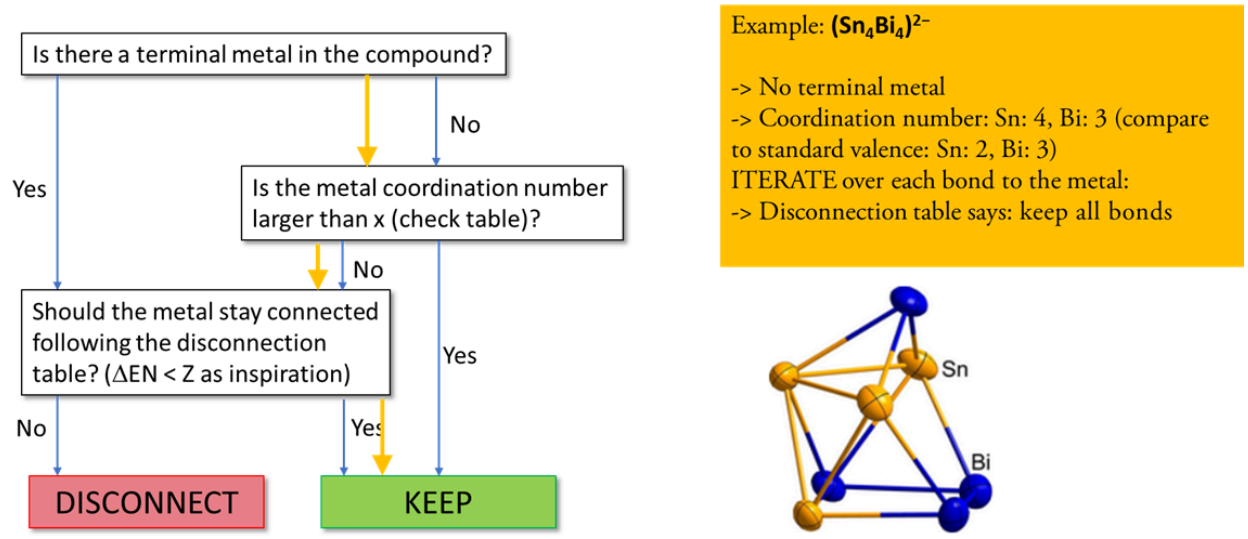


Figure S8: Bond handling for a Zintl ion synthesised by the Dehnen group[17]

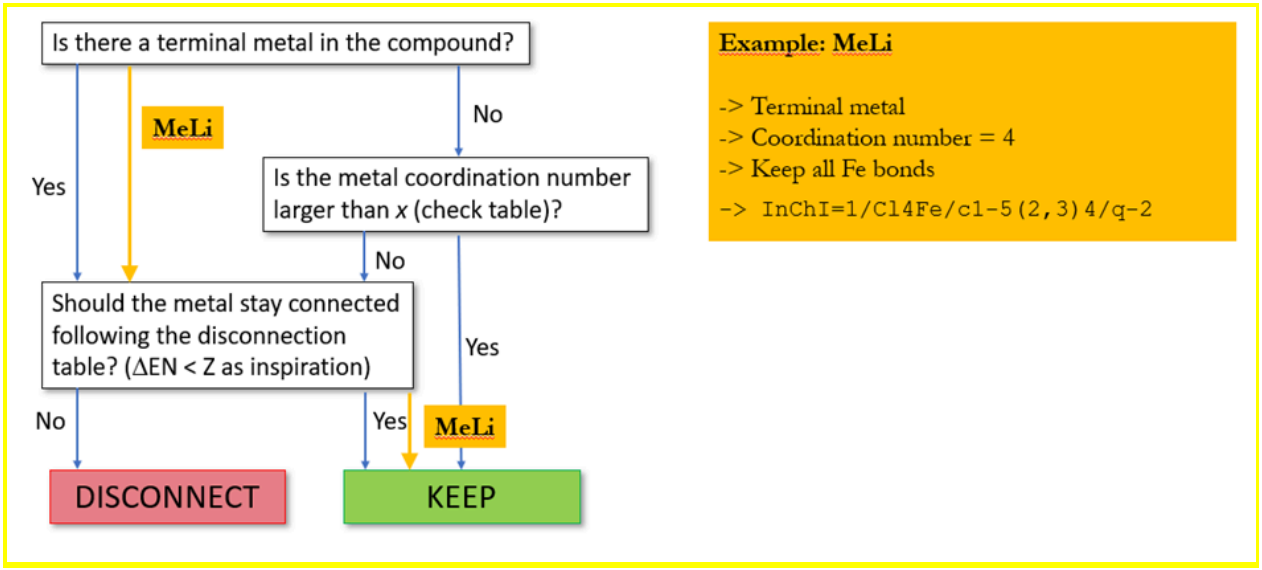


Figure S9: Bond handling for MeLi

References

[1] GitHub, *Official home of the InChI*, accessed on 17/July/2024, https://github.com/IUPAC-InChI/InChI
[2] Google, *OSS-Fuzz*, accessed on 17/July/2024, https://google.github.io/oss-fuzz/
[3] *The Fuzzing Project*, accessed on 17/July/2024, https://fuzzing-project.org/
[4] *SEI CERT C Coding Standard*, accessed on 17/July/2024, https://wiki.sei.cmu.edu/confluence/display/c/Scope
[5] *C11*, accessed on 17/July/2024, https://en.cppreference.com/w/c/11

[6] National Center for Biotechnology Information (NCBI), *PubChem Compound FTP site*, accessed on 17/July/2024, https://ftp.ncbi.nlm.nih.gov/pubchem/Compound/
[7] National Center for Biotechnology Information (NCBI), *PubChem Compound3D FTP site*, accessed on 17/July/2024, https://ftp.ncbi.nlm.nih.gov/pubchem/Compound_3D/
[8] National Center for Biotechnology Information (NCBI), *PubChem Substance FTP site*, accessed on 17/July/2024, https://ftp.ncbi.nlm.nih.gov/pubchem/Substance/
[9] InChI Trust, *InChI Web Demo*, accessed on 17/July/2024, https://iupac-inchi.github.io/InChI-Web-Demo/
[10] GitHub, *Ketcher*, accessed on 17/July/2024, https://github.com/epam/ketcher
[11] *WebAssembly*, accessed on 17/July/2024, https://webassembly.org/
[12] *Emscripten*, accessed 17/July/2024, https://emscripten.org/
[13] R. L. Apodaca, *Compiling InChI to WebAssembly Part 1: Hello InChI*, accessed on 17/July/2024, https://depth-first.com/articles/2019/05/15/compiling-inchi-to-webassembly-part-1/
[14] Kekule.js Lab, *InChI Demo*, accessed on 17/July/2024, https://partridgejiang.github.io/cheminfo-to-web/demos/items/InChI/inchiDemo.html
[15] G. Landrum and the RDKit contributors, *RDKit.js*, accessed on 17/July/2024, https://www.rdkitjs.com/
[16] GitHub, *EPAM Indigo projects*, accessed on 17/July/2024, https://github.com/epam/Indigo
[17] Y. R. Lohse, B. Weinert, B. Peerless, S. Dehnen, *Z. Anorg. Allg. Chem.* 2024, 650, e202300229