Ciccalè, M., Ferreiro, D., Jurjo-Rivas, D., Morales, J. F., Lopez-Garcia, P., Hermenegildo, M. V.: Demonstrating the Ciao Prolog Playground. In: Sáenz-Pérez, F. (ed.) Actas de las XXV Jornadas de Programación y Lenguajes (PROLE 2026). Sistedes (2026). https://hdl.handle.net/11705/PROLE/2026/20

Demonstrating the Ciao Prolog Playground*

Marco Ciccalè[1,3], Daniela Ferreiro[1,3], Daniel Jurjo-Rivas[1,3], José F. Morales[1,3], Pedro López-García[2,3], and Manuel V. Hermenegildo[1,3]

[1] Universidad Politécnica de Madrid, Madrid, Spain
[2] Spanish Council for Scientific Research, Madrid, Spain
[3] IMDEA Software Institute, Madrid, Spain
{marco.ciccale,daniela.ferreiro,daniel.jurjo}@{upm.es,imdea.org}
pedro.lopez@{csic.es,imdea.org}
{josef.morales,manuel.hermenegildo}@{upm.es,imdea.org}

Abstract. We demonstrate the Ciao Playground and the Active Logic Documents (ALDs) approach as a practical toolset designed to facilitate the teaching and learning of Prolog and formal methods, both on-line and in the classroom.

Keywords: Prolog Playground · Active Logic Documents · Teaching · Ciao Prolog · (Constraint) Logic Programming

The Ciao Prolog Playground [3,4][4] is a beginner-friendly, web-based development environment that allows users to immediately start working with the Ciao Prolog system [6], as all of its active components (editors, queries, applications, etc.) run entirely in the user's browser, without any prior local configuration nor installation. The Playground's design relies on the use of the `ciaowasm` build grade of Ciao Prolog, which compiles to WebAssembly [5] using Emscripten [16]. This is in contrast to other existing tools, which are generally server-based.[5] The Ciao Playground provides a fully integrated development environment featuring:

Top-level interaction model: Users can load, edit, and run their own Prolog files, or experiment with the available examples, through the classic and familiar top-level interface. Compilation diagnostics appear directly in the program's source code, highlighted with visual cues.

Source-level debugging: Integration with the interactive, source-level debugger, which allows stepping through the program's execution while inspecting the values of its variables, marking each execution step at the corresponding line in the program's source code.

Code sharing: Users can click on the *sharing* button (Share) that generates and copies a link to the clipboard which, when opened, launches a new

* Partially funded by MICIU projects CEX2024-001471-M *María de Maeztu* and EU GA101154447 *NEAT*. We would also like to thank the anonymous reviewers for their useful and constructive feedback.

[4] https://ciao-lang.org/playground

[5] For details on the implementation of the Ciao WebAssembly back-end (`ciaowasm`) and/or the Playground architecture, and comparison to other approaches, see [4].

```
:- module(puzzle,[solution/1]).
:- use_package(doccomments).

solution( S ) :-
 % S must be a 27 element list:
 S = [_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
 % (You can also do: N is 3*9, length(S,N))
 sublist( [1,_,1,_,1], S ),
 sublist( [2,_,_,2,_,_,2], S ),
 sublist( [3,_,_,_,3,_,_,_,3], S ),
 sublist( [4,_,_,_,_,4,_,_,_,_,4], S ),
 sublist( [5,_,_,_,_,_,5,_,_,_,_,_,5], S ),
 sublist( [6,_,_,_,_,_,_,6,_,_,_,_,_,_,6], S ),
 sublist( [7,_,_,_,_,_,_,_,7,_,_,_,_,_,_,7], S ),
 sublist( [8,_,_,_,_,_,_,_,_,8,_,_,_,_,_,_,_,8], S ),
 sublist( [9,_,_,_,_,_,_,_,_,_,9,_,_,_,_,_,_,_,_,9], S ).

%! sublist( Y, XYZ ): List `Y` is a sublist of list `XYZ`.
%! \doinclude sublist/2
sublist( Y, XYZ ) :-
 append( _X, YZ, XYZ ), append( Y, _Z, YZ ).
```

run ▶

Fig. 1. An example with an embedded link to the Playground (obtained via Share).

Playground instance with the same editor window contents. The Playground encodes the editor contents directly in the link, eliminating the need for a server and making it easy to embed "click-to-run" links to runnable code examples in teaching materials such as tutorials, manuals, slides, exercises, technical papers, notebooks, etc. For example, the run ▶ button in Fig. 1 contains a link to the Playground, which is embedded in this document, contains the program, and was obtained using the Share button.

Automatic documentation from source: Integration with the LPdoc auto-documenter [8,9], which generates documentation from the source code by means of machine-readable comments and assertions. Within the Playground, users can click on the *documentation generation* button (📖) for LPdoc to generate HTML documentation for their programs, which they can preview directly within the same Playground instance.

Advanced features: In addition to the above mentioned features, the Playground also integrates a large variety of other (more advanced) tooling. For example, the CiaoPP program preprocessor, which includes support for unit testing [12] and static analysis and verification of properties such as modes, types, determinacy, non-failure, etc. [10,11]. Again, all running locally within the user's browser; these features are available under the *more* button (⋯).

Given its modular architecture, the Playground can be easily specialized for particular applications. One example is the s(CASP) Playground, which integrates the s(CASP) goal-directed interpreter for answer set programming with constraints [1].[6] Another example is the LPTP Playground, which integrates the LPTP logic program theorem prover [15].[7]

[6] https://ciao-lang.org/playground/scasp.html
[7] https://ciao-lang.org/playground/lptp.html

Ciccalè, M., Ferreiro, D., Jurjo-Rivas, D., Morales, J. F., Lopez-Garcia, P., Hermenegildo, M. V.: Demonstrating the Ciao Prolog Playground. In: Sáenz-Pérez, F. (ed.) Actas de las XXV Jornadas de Programación y Lenguajes (PROLE 2026). Sistedes (2026). https://hdl.handle.net/11705/PROLE/2026/20

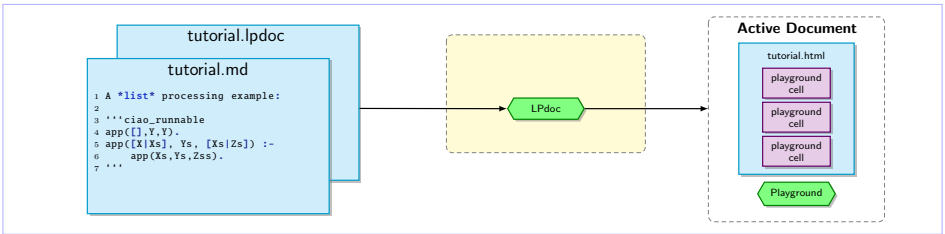

Fig. 2. Generating Active Logic Documents.

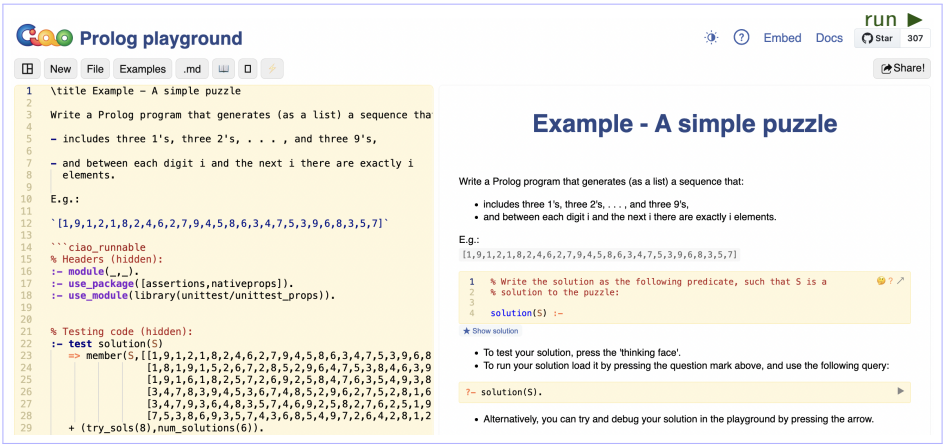

Fig. 3. Creating and editing an Active Logic Document in the Playground. Click on the "run ▶" button above to open in Playground (the link was also obtained via Share).

Active Logic Documents (ALDs) [13,14] are interactive, web-based notebooks that *embed* Prolog engines. They are composed of simple `.md` (Markdown) files in which Prolog code blocks containing examples and queries can be marked as `runnable`. These files are processed by LPdoc to generate the corresponding `.html` (HTML) documents, where `runnable` code blocks appear as embedded mini-Playground cells (see Fig. 2), allowing users to interact with the examples directly within the generated documents. Additionally, `runnable` cells can also contain interactive exercises with *hints* (to help users find the solution to a given exercise) and *solutions*, tests (for self-evaluation), non-visible parts, etc.

For example, this document[8] is an ALD containing Werner Hett's 99 Prolog problems, organized into sections (*e.g.*, lists, binary trees, etc.). Each problem includes a description, an example, and a `runnable` cell with hints.

ALDs can be used to very easily create editable and runnable educational resources, such as on-line tutorials, slides, activities, runnable examples, programming exercises, websites, manuals, etc.; even for more advanced features, such as static analysis or verification tutorials [2].

The integration of ALDs and the Playground enables the following features:

[8] The file is available at https://cliplab.org/logalg/doc/99problemsALD.html

Ciccalè, M., Ferreiro, D., Jurjo-Rivas, D., Morales, J. F., Lopez-Garcia, P., Hermenegildo, M. V.: Demonstrating the Ciao Prolog Playground. In: Sáenz-Pérez, F. (ed.) Actas de las XXV Jornadas de Programación y Lenguajes (PROLE 2026). Sistedes (2026). https://hdl.handle.net/11705/PROLE/2026/20

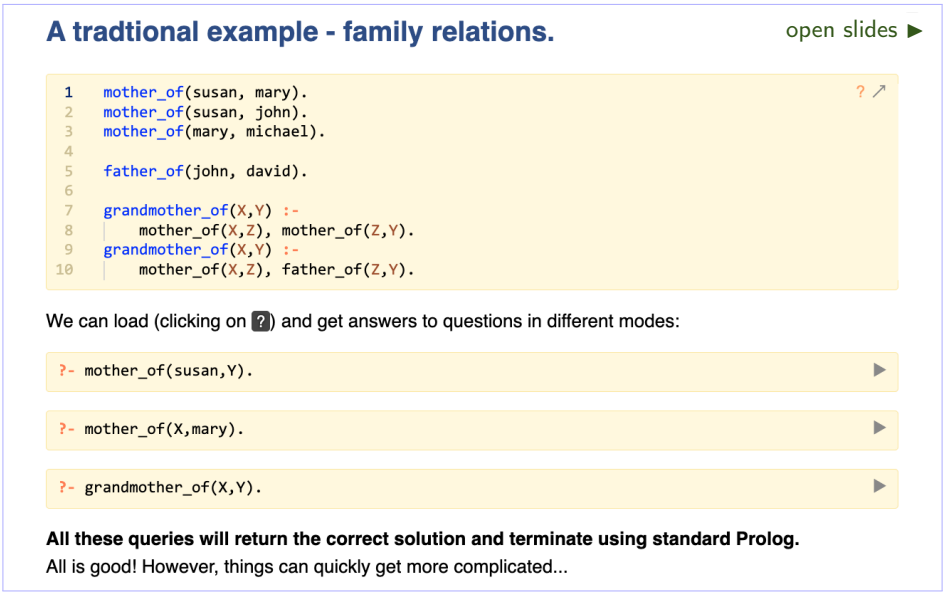

Fig. 4. An example ALD slide from a presentation. Click on the "open slides ▶" button above to open in the Playground for viewing and editing the full set of slides.

In-Playground creation and editing: Users can develop ALDs directly from the Playground, with `runnable` cells running within itself. To this end, the Playground has two distinct modes, *code mode* for editing source code (`.pl` files), and *document mode* for editing markup text (`.md` files). The currently enabled mode is indicated with the *mode switch* buttons (.md and .pl; see Fig. 3). These two modes enable a flexible and integrated workflow: in code mode, the user can write and test code snippets, and also annotate them with assertions or machine-readable comments, and in document mode, the user can incorporate these examples into documents as exercises, notes, etc.

Presentation mode: ALDs can be displayed in *presentation mode* (by clicking in the *full-screen* button ()), providing a full view of the notebook inside the Playground. That is, the document panel (*e.g.*, the right panel in Fig. 3) is visible, whereas the editing panel, buttons and menu are hidden, similarly to a statically generated ALD. However, in contrast to statically generated ALDs, users can return to the normal Playground view and keep editing the notebook. While in document mode, links obtained with the *sharing* button (Share) will enable presentation mode by default, so that they can be used directly as notes, exercises, etc. For example, instructors can prepare exercises and notes in the editor view, and then send the links to their students (or include them in an exercise sheet, slides, etc.). Then, when accessing the exercises, students would directly see the presentation view.

Slides mode: ALDs can be also used as *slides* for presentations with embedded examples and quizzes, which can then be delivered directly within the Playground. An example is these slides (a subset of the presentation in [7];

Ciccalè, M., Ferreiro, D., Jurjo-Rivas, D., Morales, J. F., Lopez-Garcia, P., Hermenegildo, M. V.: Demonstrating the Ciao Prolog Playground. In: Sáenz-Pérez, F. (ed.) Actas de las XXV Jornadas de Programación y Lenguajes (PROLE 2026). Sistedes (2026). https://hdl.handle.net/11705/PROLE/2026/20

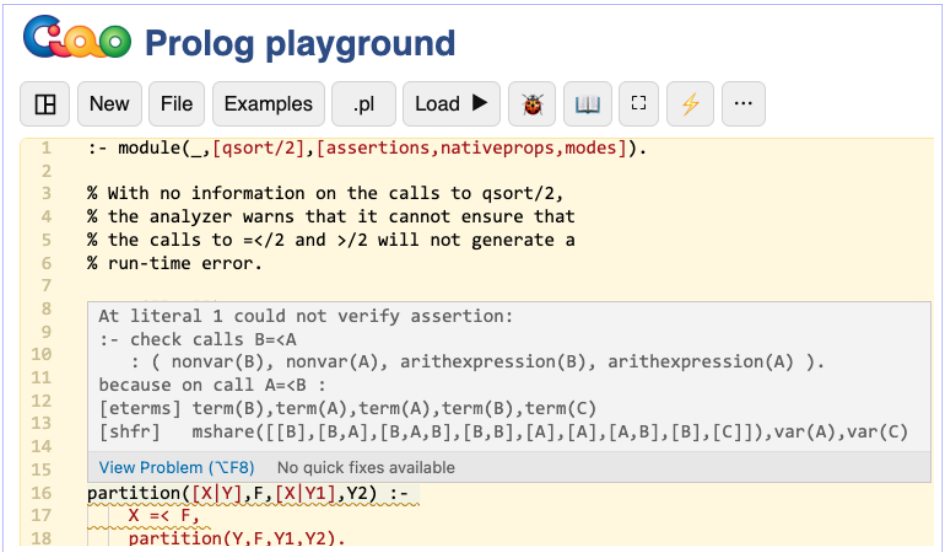

Fig. 5. Errors from static analysis appearing dynamically in on-the-fly mode ((⚡)).

see Fig. 4). The slides mode uses presentation-friendly fonts and layouts, and can be used for presenting any ALD. It is enabled by clicking on the corresponding button in the top right corner, and exited using the escape key. By default, ALDs are split into separate slides at section boundaries (other mechanisms exist, but this one is the simplest and has proved useful in practice). These presentations are full ALDs, written with the same Markdown-style syntax and allowing the inclusion of interactive components, such as code editors, query boxes, etc. This has proven particularly useful for class lectures including live demonstrations, and also workshops, etc.

The reactive (on-the-fly) mode shows feedback and results as the student types, toggled via the *on-the-fly* button ((⚡)). This includes: syntax checking, incremental goal evaluation, incremental testing, incremental static analysis and verification, incremental documentation, and, in general, dynamic result previews. Errors and warnings—from compilation, testing, static analysis, verification, etc.—are immediately marked in the source code, and partial solutions and hints are displayed as the user types. Fig. 5 shows a snapshot of how analysis results are generated dynamically as the program is being written. Similarly, the preview pane is updated dynamically as the user adds assertions or machine-readable comments to the source code or when editing an ALD.

Interactive collaboration facilities have been recently made available in the Playground, accessible through the *collaboration* button ((👥)). In particular, Playground and ALD notebook instances can be shared among several users and worked on concurrently. This is particularly useful for collaboration between the lecturer and students in the classroom (both in person and remote), and for collaborative work among students.

Ciccalè, M., Ferreiro, D., Jurjo-Rivas, D., Morales, J. F., Lopez-Garcia, P., Hermenegildo, M. V.: Demonstrating the Ciao Prolog Playground. In: Sáenz-Pérez, F. (ed.) Actas de las XXV Jornadas de Programación y Lenguajes (PROLE 2026). Sistedes (2026). https://hdl.handle.net/11705/PROLE/2026/20

References

1. Arias, J., Carro, M., Salazar, E., Marple, K., Gupta, G.: Constraint Answer Set Programming without Grounding. Theory and Practice of Logic Programming **18**(3-4), 337–354 (2018). https://doi.org/10.1017/S1471068418000285
2. Ferreiro, D., Morales, J., Abreu, S., Hermenegildo, M.: Demonstrating (Hybrid) Active Logic Documents and the Ciao Prolog Playground, and an Application to Verification Tutorials. In: Technical Communications of the 39th International Conference on Logic Programming (ICLP 2023). Electronic Proceedings in Theoretical Computer Science (EPTCS), vol. 385, pp. 324–330. Open Publishing Association (OPA) (July 2023), https://cliplab.org/papers/hald-demo-iclp-tc.pdf, see also associated poster at https://cliplab.org/papers/hald-poster-iclp.pdf
3. Garcia-Pradales, G., Morales, J., Hermenegildo, M.: The Ciao Prolog Playground. Tech. rep., Technical University of Madrid (UPM) and IMDEA Software Institute (2021), https://ciao-lang.org/ciao/build/doc/ciao_playground.html/ciao_playground_manual.html
4. Garcia-Pradales, G., Morales, J., Hermenegildo, M., Arias, J., Carro, M.: An s(CASP) In-Browser Playground based on Ciao Prolog. In: Proceedings of the 38th ICLP Workshops - Goal-directed Execution of Answer Set Programs. vol. 3193. CEUR-WS.org (August 2022), https://ceur-ws.org/Vol-3193/short3GDE.pdf
5. Haas, A., Rossberg, A., Schuff, D.L., Titzer, B.L., Holman, M., Gohman, D., Wagner, L., Zakai, A., Bastien, J.F.: Bringing the web up to speed with webassembly. In: Cohen, A., Vechev, M.T. (eds.) Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017. pp. 185–200. ACM (2017). https://doi.org/10.1145/3062341.3062363, https://doi.org/10.1145/3062341.3062363
6. Hermenegildo, M.V., Bueno, F., Carro, M., Lopez-Garcia, P., Mera, E., Morales, J., Puebla, G.: An Overview of Ciao and its Design Philosophy. Theory and Practice of Logic Programming **12**(1–2), 219–252 (January 2012). https://doi.org/10.1017/S1471068411000457, https://arxiv.org/abs/1102.5497
7. Hermenegildo, M.V., Morales, J.F., Lopez-Garcia, P.: Teaching Pure LP with Prolog and a Fair Search Rule. In: Proceedings of the 40th ICLP Workshops. vol. 3799. CEUR-WS.org (October 2024), https://ceur-ws.org/Vol-3799/paper2PEG2.0.pdf
8. Hermenegildo, M.: A Documentation Generator for (C)LP Systems. In: International Conference on Computational Logic, CL2000. pp. 1345–1361. No. 1861 in LNAI, Springer-Verlag (July 2000)
9. Hermenegildo, M., Morales, J.: The LPdoc Documentation Generator. Ref. Manual (v3.0). Tech. rep., UPM (July 2011), available at https://ciao-lang.org
10. Hermenegildo, M., Morales, J., Lopez-Garcia, P., Carro, M.: Types, modes and so much more – the Prolog way. In: Warren, D.S., Dahl, V., Eiter, T., Hermenegildo, M.V., Kowalski, R., Rossi, F. (eds.) Prolog - The Next 50 Years, chap. 2, pp. 23–37. No. 13900 in LNCS, Springer (July 2023), https://cliplab.org/papers/AssertionsAndOther-PrologBook.pdf
11. Hermenegildo, M., Puebla, G., Bueno, F., Lopez-Garcia, P.: Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). Science of Computer Programming **58**(1–2), 115–140 (October 2005). https://doi.org/10.1016/j.scico.2005.02.006
12. Mera, E., Lopez-Garcia, P., Hermenegildo, M.: Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework. In: 25th Int'l. Conference on Logic Programming (ICLP'09). LNCS, vol. 5649, pp. 281–295. Springer-Verlag (July 2009)

Ciccalè, M., Ferreiro, D., Jurjo-Rivas, D., Morales, J. F., Lopez-Garcia, P., Hermenegildo, M. V.: Demonstrating the Ciao Prolog Playground. In: Sáenz-Pérez, F. (ed.) Actas de las XXV Jornadas de Programación y Lenguajes (PROLE 2026). Sistedes (2026). https://hdl.handle.net/11705/PROLE/2026/20

13. Morales, J., Abreu, S., Ferreiro, D., Hermenegildo, M.: Teaching Prolog with Active Logic Documents. Tech. Rep. CLIP-1/2022.0, Technical University of Madrid (UPM) and IMDEA Software Institute (December 2022)
14. Morales, J., Abreu, S., Ferreiro, D., Hermenegildo, M.: Teaching Prolog with Active Logic Documents. In: Warren, D.S., Dahl, V., Eiter, T., Hermenegildo, M.V., Kowalski, R., Rossi, F. (eds.) Prolog - The Next 50 Years, chap. 14, pp. 171–183. No. 13900 in LNCS, Springer (July 2023), https://cliplab.org/papers/ActiveLogicDocuments-PrologBook.pdf
15. Stärk, R.F.: The theoretical foundations of LPTP (A logic program theorem prover). J. Log. Program. **36**(3), 241–269 (1998). https://doi.org/10.1016/S0743-1066(97)10013-9, https://doi.org/10.1016/S0743-1066(97)10013-9
16. Zakai, A.: Emscripten: an LLVM-to-JavaScript compiler. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications. pp. 301–312. SPLASH '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/2048147.2048224, http://doi.acm.org/10.1145/2048147.2048224