Some Enhancements to the Prolog Playground and ALD Notebooks for the Modern Prolog Classroom*

Jose F. Morales[1,2], Daniela Ferreiro[1,2], Marco Pérez[1,2] and Manuel V. Hermenegildo[1,2]

[1]*Universidad Politécnica de Madrid (UPM), Madrid, Spain*
[2]*IMDEA Software Institute, Madrid, Spain*

Abstract
One of the main objectives of the Prolog Playground and ALD-style notebooks (Active Logic Documents) is to facilitate the task of teaching and learning Prolog. The Playground allows teachers and students to develop, run, debug, verify, and test Prolog programs directly on the browser, without needing a Prolog installation. It also greatly facilitates the sharing of Prolog code and adding click-to-run capabilities to any kind of teaching materials. ALD-style notebooks further facilitate the generation of web-based materials with embedded running examples and exercises. Fundamental to these tools is that they run locally on the student's browser, with no need for a server infrastructure. We will report on (and demo) some recent important extensions which include a) being able to develop the notebooks with embedded running examples and quizzes now directly within the Playground, b) also being able to develop *slides* with such examples and quizzes as well as doing presentations within the Playground, c) a fully reactive mode where feedback and results are given on-the-fly, as the student types, d) new facilities for interactive collaboration in the classroom, and e) integration with the LPTP theorem prover. We hope these facilities will be of interest to teachers and students of the Prolog language.

Keywords
Active Logic Documents, Prolog Playground, Prolog Notebooks, Teaching Prolog, Prolog, Ciao Prolog, Logic Programming, WebAssembly.

1. Introduction

The Prolog Playground [1, 2] [1] and ALD-style notebooks (Active Logic Documents) [3] constitute a toolset designed to facilitate the task of teaching and learning Prolog, both online and in the classroom. An important characteristic of these tools, is that everything (editors, queries, applications, etc.) runs locally on the student's browser, with no need for a server infrastructure. This is based on the use of the `ciaowasm` build grade of Ciao Prolog (i.e., using WebAssembly [4] compiled with Emscripten [5]). This is in contrast to other tools which are generally server-based.[2] After providing a brief overview of the previous characteristics of this toolset in Section 2, our purpose herein is to report on (and demo) some recent extensions to these tools, presented in Section 3.

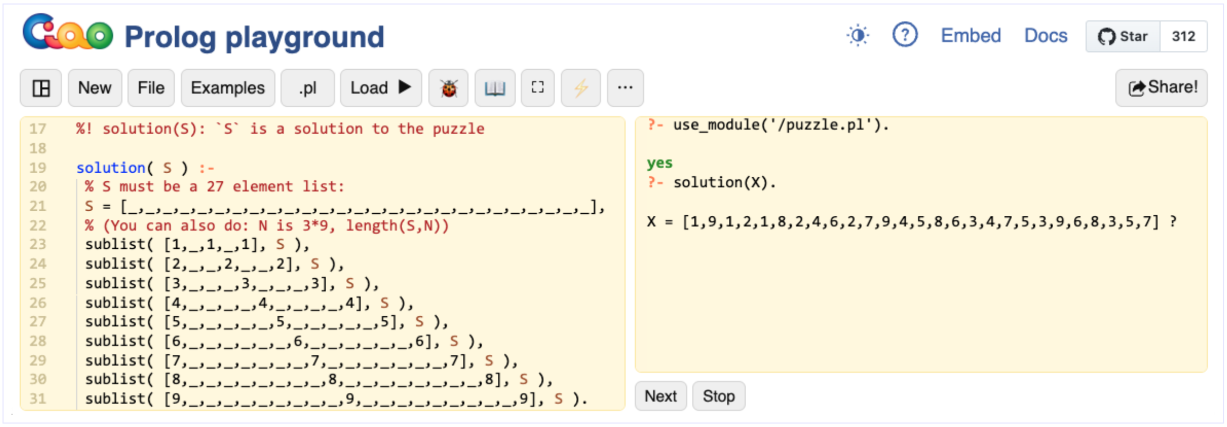
2. Previous Playground and ALD Characteristics

The Prolog Playground: Since its first versions, the first component of the toolset, the *Prolog Playground*, allows teachers and students to perform a number of tasks directly within the browser, providing essentially a full IDE, without any need for a Prolog installation or being connected to a server. These include:

- Code can be loaded from the user's local files or the web, edited, run and debugged using the classic and familiar top-level interaction model (see Fig. 1), and saved, all locally on the user's web

*Partially funded by MICIU projects CEX2024-001471-M *María de Maeztu* and TED2021-132464B-I00, by EU Grant 101154447 NEAT, and by the Tezos foundation.
Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

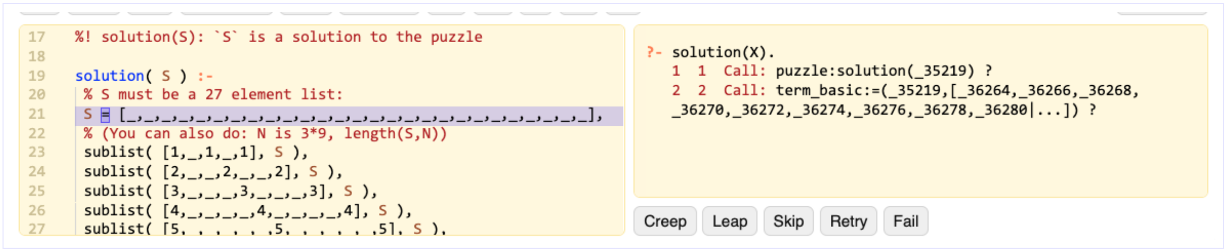
[1]http://ciao-lang.org/playground
[2]See [3] for a more complete discussion and references.



```
%! solution(S): `S` is a solution to the puzzle

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 ).
```

```
?- use_module('/puzzle.pl').

yes
?- solution(X).

X = [1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7] ?
```

Figure 1: Editing and running code (New File Examples .pl Load ▶).



```
%! solution(S): `S` is a solution to the puzzle

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 ),
```

```
?- solution(X).
   1  1  Call: puzzle:solution(_35219) ?
   2  2  Call: term_basic:=(_35219,[_36264,_36266,_36268,
   _36270,_36272,_36274,_36276,_36278,_36280|...]) ?
```

Figure 2: Source-level debugging (🐞).

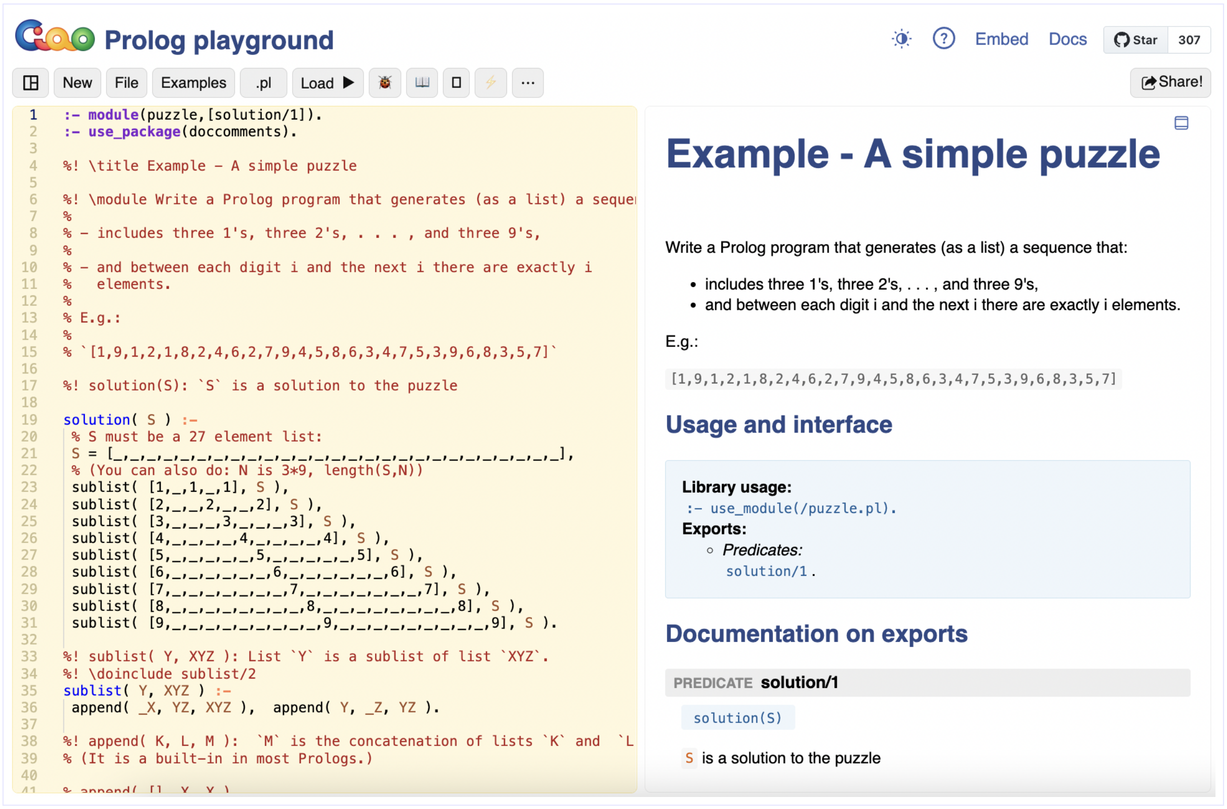
run ▶

```
:- 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 ).
```

Figure 3: An example with an embedded link to the Playground (obtained via Share).

browser. The debugger allows stepping through the code in the traditional way, with the position (as well as errors, etc.) being marked on the source in the editor window (Fig. 2). Solutions to goals can be portrayed in ways other than just printing them out, allowing output of graphical and user-interface components. An example is giving explanations in the s(CASP) Playground (see later).

- Code sharing: clicking on the Share button, a link is generated that opens a Playground with the contents of the editor window (see this link and Fig. 3 for a LaTeX example; note that the link with the code is stored within this document and has not been uploaded to any server). This link can be mailed or pasted into applications or documents. This allows very easily embedding "click-to-run" links to runnable code examples in any kind of teaching materials, including tutorials, manuals, slides, exercises, technical papers, notebooks, etc., developed with any tool.



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

%! \title Example - A simple puzzle

%! \module Write a Prolog program that generates (as a list) a seque
%
% - includes three 1's, three 2's, . . . , and three 9's,
%
% - and between each digit i and the next i there are exactly i
%   elements.
%
% E.g.:
%
% `[1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7]`

%! solution(S): `S` is a solution to the puzzle

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 ).

%! append( K, L, M ):  `M` is the concatenation of lists `K` and  `L
% (It is a built-in in most Prologs.)

% append( [], X, X )
```

Example - A simple puzzle

Write a Prolog program that generates (as a list) a sequence that:

- includes three 1's, three 2's, . . . , and three 9's,
- and between each digit i and the next i there are exactly i elements.

E.g.:

`[1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7]`

Usage and interface

Library usage:
`:- use_module(/puzzle.pl).`
Exports:
- *Predicates:*
  `solution/1.`

Documentation on exports

PREDICATE **solution/1**

`solution(S)`

`S` is a solution to the puzzle

Figure 4: Generating program documentation (📖).

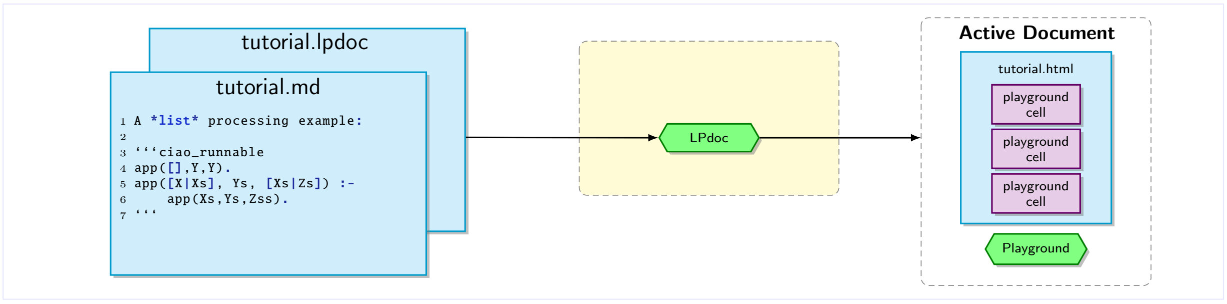
- Developing Prolog code documentation: the `LPdoc` documentation generator [6, 7] is also embedded within the Playground. This allows generating and previewing documentation obtained from the source code itself and any machine-readable comments and/or assertions, also locally within the browser (📖 button; Fig. 4).

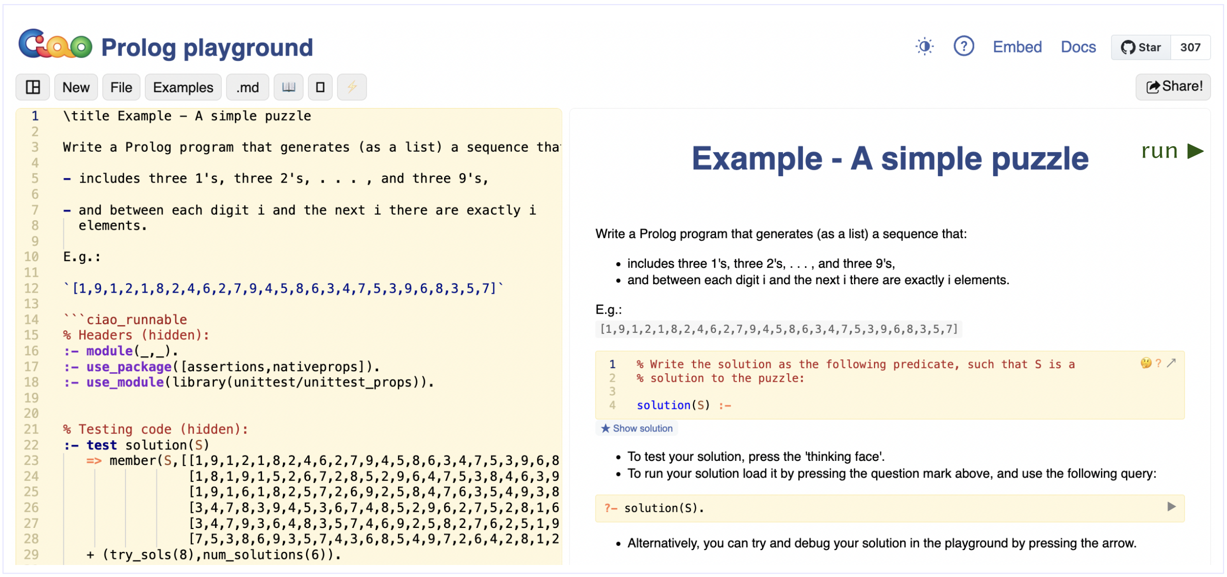
- Advanced features (··· button) include running any unit tests [8] that may be present in the program; analyzing and verifying properties such as modes, types, determinacy, non-failure, etc. [9, 10]; partial evaluation and code specialization; etc., again all running locally within the browser.

The Playground can be easily specialized for particular applications, an example of which is the s(CASP) Playground [2] [3] or the LPTP Playground (see later). More information on the implementation of the Ciao WebAssembly back-end and the Playground architecture can be found in [2].

Active Logic Documents: The other main component of the toolset is *Active Logic Documents* (ALDs) [3]. ALDs are notebooks which incorporate *embedded* Prolog engines. For example, this document[4] is an ALD containing Werner Hett's 99 Prolog problems encoded as executable exercises. ALDs are composed from simple `.md` (markdown) files, where examples and queries can be marked as `runnable`. These are then included in the document in embedded Prolog mini-Playground cells, so that interaction with these examples is possible within the documents produced. The actual generation of the html files is performed by `LPdoc` (Fig. 5). Runnable cells can also contain interactive exercises with hints (to help students find a solution) and solutions, tests (for self-evaluation of the student-provided solutions), non-visible parts, etc. (as in the 99 Prolog problems example). There is also an escape mechanism to javascript that allows easily implementing additional functionality, graphical output, etc. (`jseval`) and ALD fragments can be dynamically generated from programs (`dynpreview`). ALDs can

[3]https://ciao-lang.org/playground/scasp.html
[4]https://cliplab.org/logalg/doc/99problemsALD.html/



~~~
A *list* processing example:

```ciao_runnable
app([],Y,Y).
app([X|Xs], Ys, [Xs|Zs]) :-
    app(Xs,Ys,Zss).
~~~
```

Figure 5: Generating Active Logic Documents.


Prolog playground
New
File
Examples
.md
Embed
Docs
Star
307
Share!
run ▶
★ Show solution

```
\title Example - A simple puzzle

Write a Prolog program that generates (as a list) a sequence tha

- includes three 1's, three 2's, . . . , and three 9's,

- and between each digit i and the next i there are exactly i
  elements.

E.g.:

`[1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7]`

```ciao_runnable
% Headers (hidden):
:- module(_,_).
:- use_package([assertions,nativeprops]).
:- use_module(library(unittest/unittest_props)).


% Testing code (hidden):
:- test solution(S)
   => member(S,[[1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8
                [1,8,1,9,1,5,2,6,7,2,8,5,2,9,6,4,7,5,3,8,4,6,3,9
                [1,9,1,6,1,8,2,5,7,2,6,9,2,5,8,4,7,6,3,5,4,9,3,8
                [3,4,7,8,3,9,4,5,3,6,7,4,8,5,2,9,6,2,7,5,2,8,1,6
                [3,4,7,9,3,6,4,8,3,5,7,4,6,9,2,5,8,2,7,6,2,5,1,9
                [7,5,3,8,6,9,3,5,7,4,3,6,8,5,4,9,7,2,6,4,2,8,1,2
   + (try_sols(8),num_solutions(6)).
```

Example - A simple puzzle

Write a Prolog program that generates (as a list) a sequence that:

- includes three 1's, three 2's, . . . , and three 9's,
- and between each digit i and the next i there are exactly i elements.

E.g.:
`[1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7]`

```
% Write the solution as the following predicate, such that S is a
% solution to the puzzle:

solution(S) :-
```

- To test your solution, press the 'thinking face'.
- To run your solution load it by pressing the question mark above, and use the following query:

```
?- solution(S).
```

- Alternatively, you can try and debug your solution in the playground by pressing the arrow.

Figure 6: Creating and Editing an Active Logic Document in the Playground. Click on 'run ▶' above to open in Playground (the link was also obtained via Share).

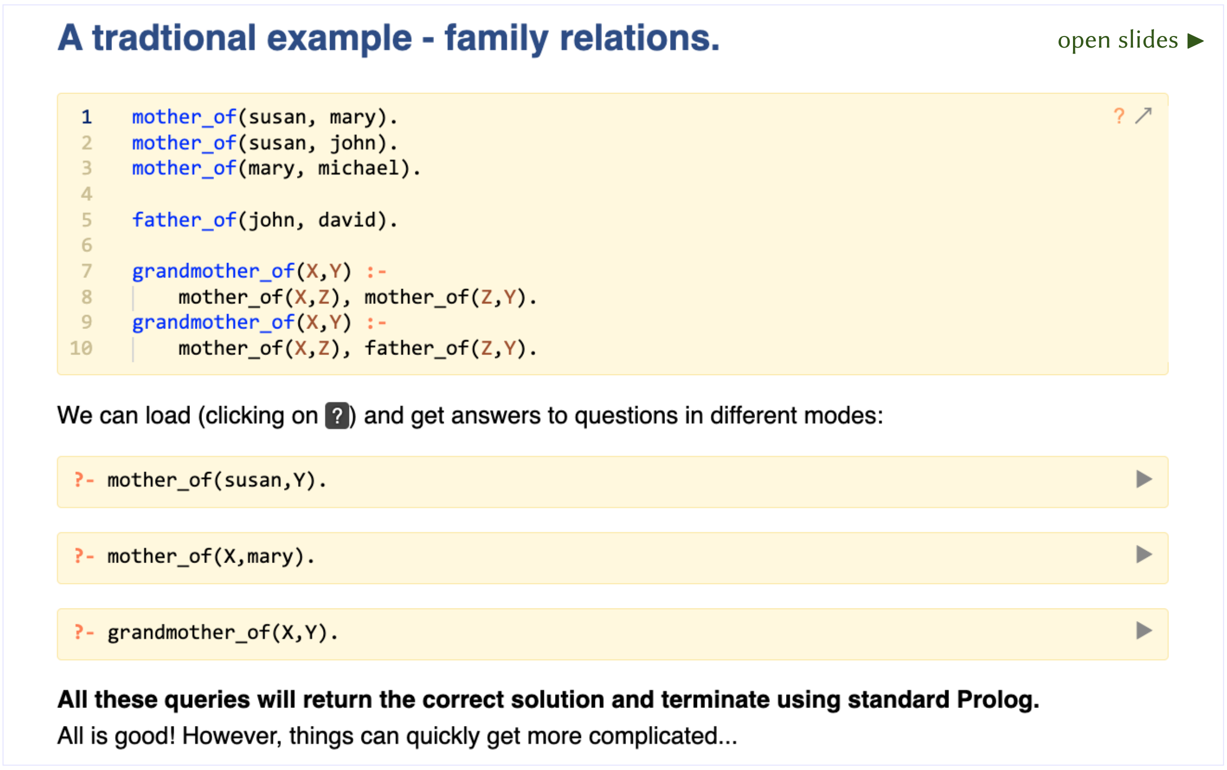
be used to very easily create editable and runnable educational resources for teaching Prolog and logic programming, such as on-line tutorials, slides, activities, runnable examples, programming exercises, web sites, manuals, etc. It is also possible to develop tutorials for advanced features, such as analyzers and verifiers [11].

In comparison with other approaches to notebooks [12, 13, 14, 15], a fundamental aspect of the ALD approach is that all the reactive parts run locally on the user's web browser without any dependency on a central server or a local Prolog installation. Also, output can be generated from and stored in simple source files that can be developed with any editor. Arguably, the ALD approach has advantages from the point of view of scalability, low maintenance cost, security, privacy, ease of packaging and distribution, etc.

3. Some New Features

We now turn to describing some new extensions and improvements to the Prolog Playground and ALD-style notebooks since the first versions of [2, 3] that we described in the previous section:

In-Playground creation and editing of Active Logic Documents. One of the main improvements since the early versions has been to better support *within the Playground* the process of developing ALD notebooks with embedded running examples and quizzes. To this end the Playground has two distinct modes, *code mode* and *document mode* (see Figure 6). The current mode is identified by the .md or .pl button shown in the button bar, which also allows changing the mode (Figure 6). You can see an example by clicking on 'run ▶' in Figure 6. The ALD can then be edited by clicking on the pencil icon on the top right. The link was obtained as usual with the Share button. These two modes



```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).

father_of(john, david).

grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

We can load (clicking on ?) and get answers to questions in different modes:

```
?- mother_of(susan,Y).
```

```
?- mother_of(X,mary).
```

```
?- grandmother_of(X,Y).
```

All these queries will return the correct solution and terminate using standard Prolog.

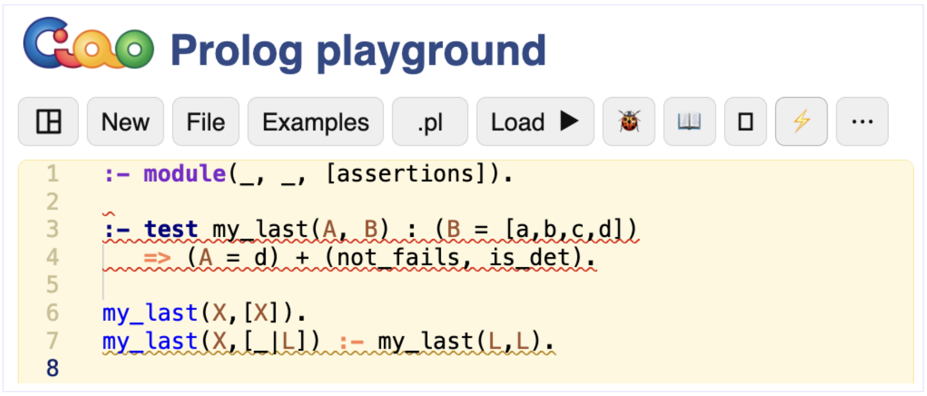
All is good! However, things can quickly get more complicated...

Figure 7: An example ALD slide from a presentation. Click on 'open slides ▶' above to open in the Playground for viewing and editing the full set of slides.

enable a flexible and integrated workflow: in code mode one can write and test code snippets, and also annotate them with documentation and explanations, and in document mode one can incorporate these examples into documents as exercises, notes, etc., all fully within the browser.

Presentation mode. The mode in which the ALD notebook appears when clicking on 'run ▶' in Figure 6 is another improvement: *presentation mode*. Presentation mode provides the traditional full view of ALD notebooks but within the Playground. I.e., in presentation mode, only the document panel (the right panel in Figure 6) is visible, without buttons or menus, similarly to a statically generated ALD. However, in contrast to statically generated ALDs, by clicking on the pencil icon ✎ on the top right, one can enter (or return to) the normal Playground to view and edit the ALD notebook. Presentation mode can also be entered from the editing view via the ⛶ button. Links obtained with the Share button in document mode will open by default in presentation mode 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 students (or include them in an exercise sheet, slides, etc.), and the students see the presentation view when they click on these links.

ALD slides. A further improvement is the possibility of creating ALDs as slides for presentations (still with embedded examples and quizzes), and of delivering the presentations directly with the Playground. An example is 'these slides' (a subset of the presentation of [16]; see Figure 7). Any ALD can be presented in slides form by clicking on the rectangular icon top right ▭, which switches modes. By default in slides mode the ALD is split into separate slides at section boundaries (other mechanisms exist, but this one is the simplest and has proved useful in practice). Slides mode also uses presentation-friendly fonts and layout. Slides mode can be exited using the escape key and then the pencil icon can be used to edit the slides in the standard way. And again the Share button can be used to share the slides. 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.

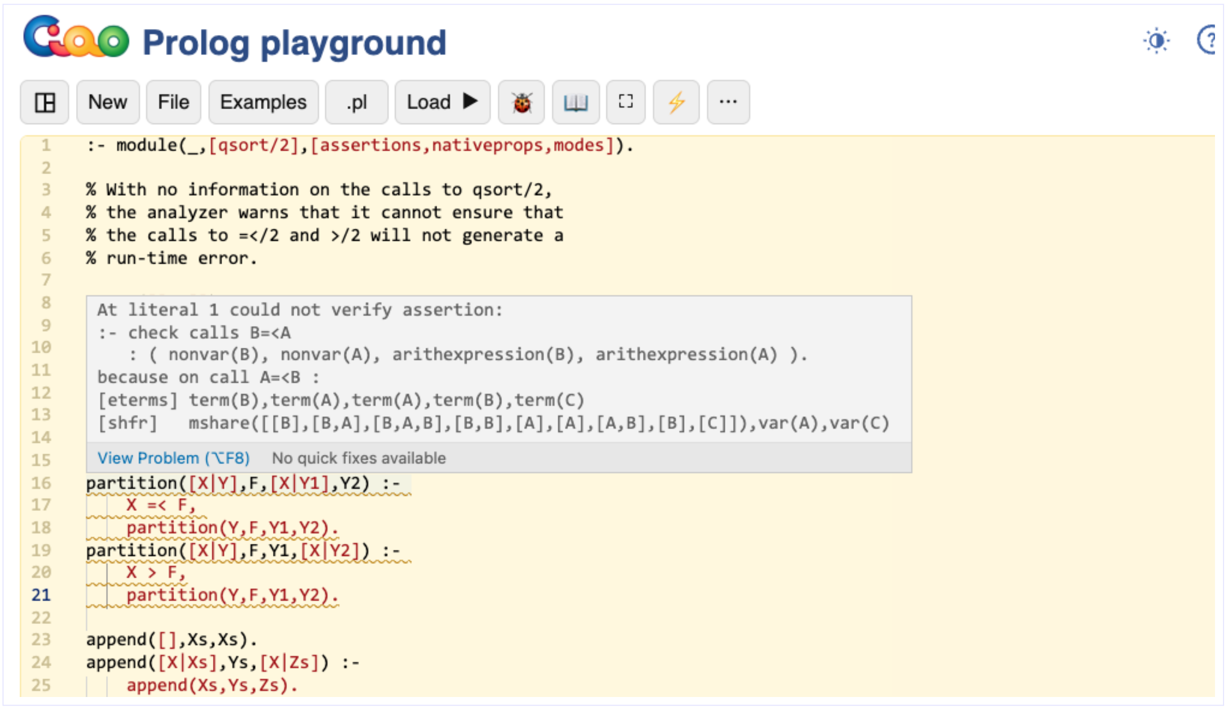


```
1  :- module(_, _, [assertions]).
2
3  :- test my_last(A, B) : (B = [a,b,c,d])
4      => (A = d) + (not_fails, is_det).
5
6  my_last(X,[X]).
7  my_last(X,[_|L]) :- my_last(L,L).
8
```

Figure 8: Errors from tests appearing dynamically in on-the-fly mode (⚡).

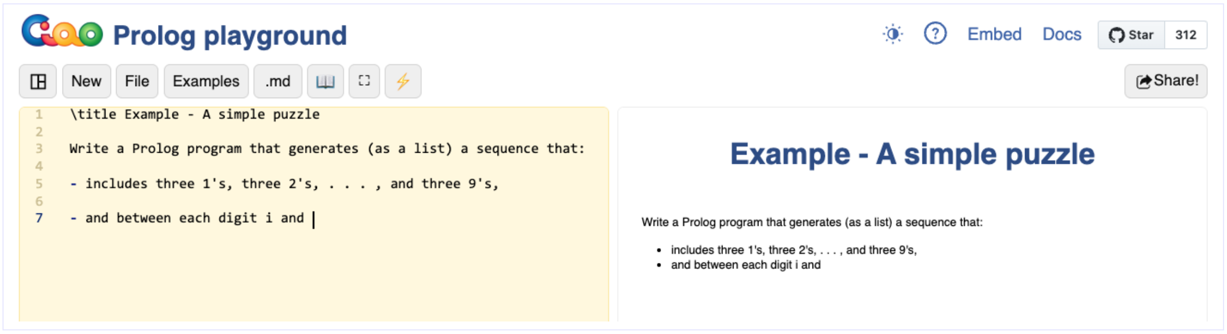
Reactive (on-the-fly) mode. A fully reactive mode is now available (toggled via the ⚡ button), in which feedback and results are shown on-the-fly as the student types. This includes real-time syntax checking, incremental goal evaluation, incremental testing, incremental analysis/verification, incremental documentation, and in general dynamic result previews. As a result, errors and warnings (from compilation, testing, static analysis/verification, etc.) are immediately highlighted on the source, and partial solutions and hints displayed, as the code is typed. As a simple example, Figure 8 shows a situation in which an error is introduced (in line 7), the tests are automatically triggered, and a failing test is underlined in red, indicating on the fly that the change causes that test to not pass. Figure 9 is a snapshot of how analysis results are also generated dynamically as the program is typed. Also, documentation is generated and the preview pane updated dynamically as the comments or assertions are typed in the code pane. The same applies to ALD notebooks, which are generated and visualized incrementally in the visualization pane. Figure 10 shows a snapshot of an ALD being generated dynamically in on-the-fly mode as the markdown source is being typed.

Interactive collaboration. New facilities have been added for this purpose (👥 button). In particular, Prolog Playground and ALD notebook instances can now 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.



```
:- module(_,[qsort/2],[assertions,nativeprops,modes]).

% With no information on the calls to qsort/2,
% the analyzer warns that it cannot ensure that
% the calls to =</2 and >/2 will not generate a
% run-time error.

 At literal 1 could not verify assertion:
 :- check calls B=<A
     : ( nonvar(B), nonvar(A), arithexpression(B), arithexpression(A) ).
 because on call A=<B :
 [eterms] term(B),term(A),term(A),term(B),term(C)
 [shfr]   mshare([[B],[B,A],[B,A,B],[B,B],[A],[A],[A,B],[B],[C]]),var(A),var(C)

 View Problem (⌥F8)   No quick fixes available
partition([X|Y],F,[X|Y1],Y2) :-
    X =< F,
    partition(Y,F,Y1,Y2).
partition([X|Y],F,Y1,[X|Y2]) :-
    X > F,
    partition(Y,F,Y1,Y2).

append([],Xs,Xs).
append([X|Xs],Ys,[X|Zs]) :-
    append(Xs,Ys,Zs).
```

Figure 9: Errors from static analysis appearing dynamically in on-the-fly mode (⚡).



```
\title Example - A simple puzzle

Write a Prolog program that generates (as a list) a sequence that:

- includes three 1's, three 2's, . . . , and three 9's,

- and between each digit i and
```

Example - A simple puzzle

Write a Prolog program that generates (as a list) a sequence that:

- includes three 1's, three 2's, . . . , and three 9's,
- and between each digit i and

Figure 10: ALDs being rendered incrementally in on-the-fly mode.

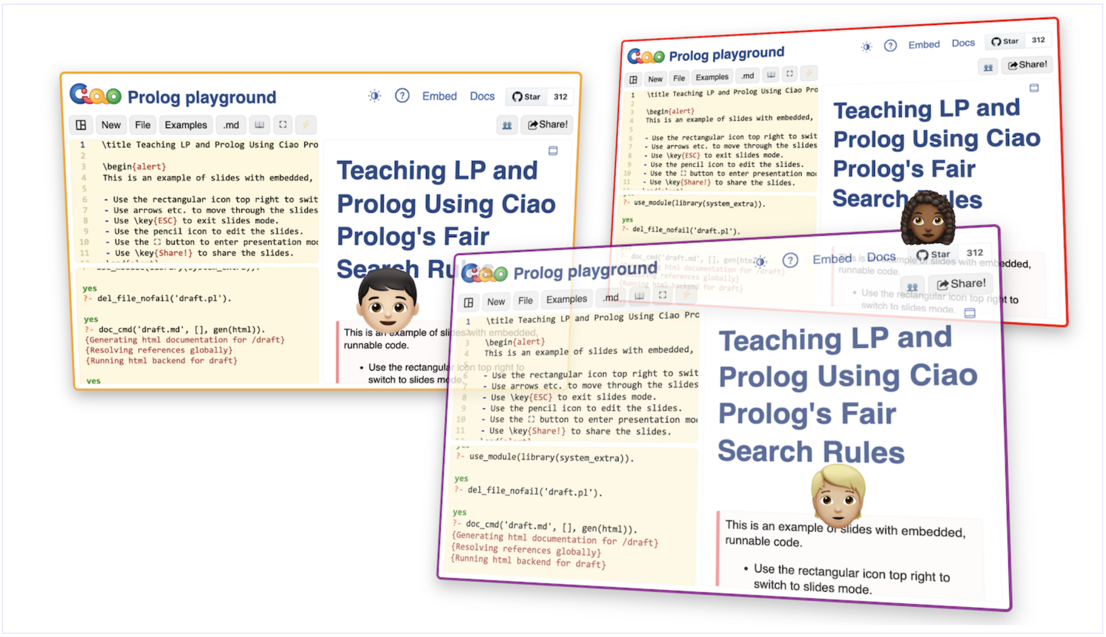
Integration of the LPTP prover. A recent extension is the integration of the LPTP theorem prover for Prolog [17] within the Playground, which has been completed in collaboration with Fred Mesnard and colleagues. Figure 12 shows an example program and part of a lemma and its proof, as well as the menu with entries for checking the proofs, choosing tactics, etc. It is also possible to include programs with proofs and run the prover within runnable cells in ALD notebooks. Space limitations do not allow us to go into more details about this very interesting topic, but this integration opens the door to teaching within the Playground and ALDs how to carry out theorem proving tasks for Prolog programs, and develop active materials, exercises, assignments, etc.

4. Conclusions: The Tools in Classroom Practice

We conclude with some comments on the use of these tools in practice in and for the classroom.

Regarding teaching materials, some lecturers and course creators often prefer to develop (or have already developed) these with their favorite tools (LaTeX, Word, PowerPoint, Keynote, Pages, etc.), and may be reluctant to use a new platform. The Playground "click-to-run" functionality is ideal here: examples in course materials can be loaded into the Playground and links generated which can then be included in these materials next to the examples, independently of the tools used or the generated format (pdf, html, etc.). Code remains in the teaching materials themselves with no dependencies on any code server. Examples of this approach can be found in the pdfs labeled "slides with embedded code" from our (C)LP courses.[5] However, for new materials or for renewing existing materials, ALDs are

[5]See https://cliplab.org/logalg.



```
\title Teaching LP and Prolog Using Ciao Pro

\begin{alert}
This is an example of slides with embedded,

- Use the rectangular icon top right to swit
- Use arrows etc. to move through the slides
- Use \key{ESC} to exit slides mode.
- Use the pencil icon to edit the slides.
- Use the ⛶ button to enter presentation mod
- Use \key{Share!} to share the slides.
```

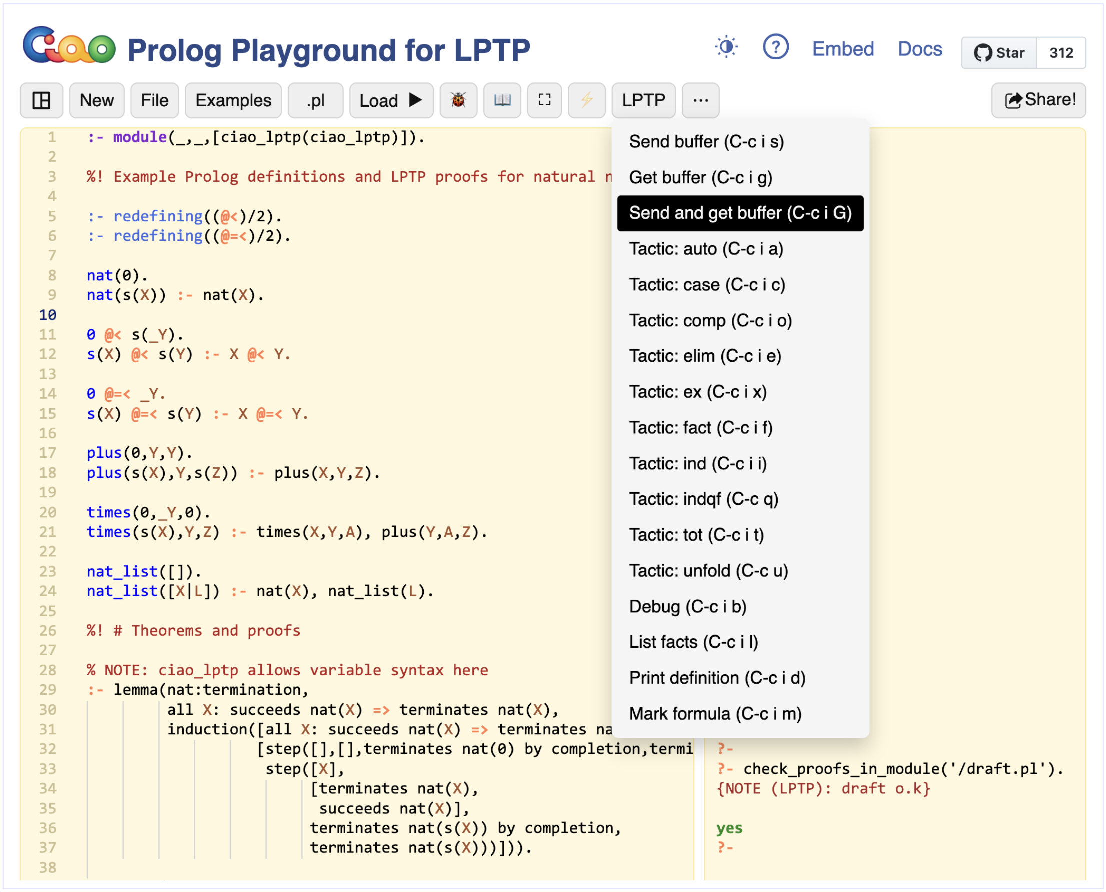
```
?- use_module(library(system_extra)).

yes
?- del_file_nofail('draft.pl').

yes
?- doc_cmd('draft.md', [], gen(html)).
{Generating html documentation for /draft}
{Resolving references globally}
{Running html backend for draft}
```

Teaching LP and Prolog Using Ciao Prolog's Fair Search Rules

This is an example of slides with embedded, runnable code.

- Use the rectangular icon top right to switch to slides mode.

Figure 11: Collaboration mode: several playgrounds connected.



```
:- module(_,_,[ciao_lptp(ciao_lptp)]).

%! Example Prolog definitions and LPTP proofs for natural n

:- redefining((@<)/2).
:- redefining((@=<)/2).

nat(0).
nat(s(X)) :- nat(X).

0 @< s(_Y).
s(X) @< s(Y) :- X @< Y.

0 @=< _Y.
s(X) @=< s(Y) :- X @=< Y.

plus(0,Y,Y).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).

times(0,_Y,0).
times(s(X),Y,Z) :- times(X,Y,A), plus(Y,A,Z).

nat_list([]).
nat_list([X|L]) :- nat(X), nat_list(L).

%! # Theorems and proofs

% NOTE: ciao_lptp allows variable syntax here
:- lemma(nat:termination,
        all X: succeeds nat(X) => terminates nat(X),
        induction([all X: succeeds nat(X) => terminates na
                  [step([],[],terminates nat(0) by completion,termi
                   step([X],
                       [terminates nat(X),
                        succeeds nat(X)],
                       terminates nat(s(X)) by completion,
                       terminates nat(s(X)))])).

```

```
?-
?- check_proofs_in_module('/draft.pl').
{NOTE (LPTP): draft o.k}

yes
?-
```

Figure 12: Integration of the LPTP prover in the Playground.

very useful, since they allow developing engaging, interactive exercises and assignments, class notes, tutorials, etc. with built-in self-assessment. The new ALD slides are also instrumental for developing material for lectures containing runnable examples. Finally, the new interactive collaboration facilities greatly facilitate involving students during lectures, keeping them engaged.

Regarding the personal work of students, we have observed that some of our (college-level, CS) students[6] still do install the Prolog system in their laptops and use VSC or Emacs, as we actually suggest, since this is always closer to how a programming language is used in application development. But the majority of students prefer the Playground, because it saves them installation time and hassle. Also, the Playground is clearly preferable in other contexts such as younger, non CS, no laptop, etc. students, The Playground and the runnable cells in ALDs offer performance that is competitive with native Prolog implementations (around 50% of Ciao Prolog native speed) and provide access to most of Ciao Prolog's functionality, including Pure LP (with several search rules), ISO-Prolog, higher-order, functional syntax, constraints, imperative syntax, ASP/s(CASP), etc. Fair search rules in particular are very useful in the early stages of LP instruction for running examples under a pure, declarative view. This avoids having to face non-termination prematurely and force introducing operational aspects too early, which can hamper the first learning steps (see [16] and [18] for arguments in favor of this approach).

The new features that we have presented are aimed at further lowering the barrier to creating and delivering interactive, browser-based, server-independent Prolog material. At the same time, they help improve the student learning experience by making it more active, responsive, and self-directed. We believe these developments make the Playground+ALDs toolset particularly well-suited for modern logic programming education. We hope these new additions to the Playground and ALDs, as well as our reflections, will be of interest to the PEG audience and in general to all those interested in Prolog Education.

[6]We teach a full semester of Prolog/(C)LP to upwards of 400 CS undergraduate and graduate students every year.

References

[1] G. Garcia-Pradales, J. Morales, M. Hermenegildo, The Ciao Prolog Playground, Technical Report, Technical University of Madrid (UPM) and IMDEA Software Institute, 2021. URL: https://ciao-lang.org/ciao/build/doc/ciao_playground.html/ciao_playground_manual.html.

[2] G. Garcia-Pradales, J. Morales, M. Hermenegildo, J. Arias, M. Carro, An s(CASP) In-Browser Playground based on Ciao Prolog, in: Proceedings of the 38th ICLP Workshops - Goal-directed Execution of Answer Set Programs, volume 3193, CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3193/short3GDE.pdf.

[3] J. Morales, S. Abreu, D. Ferreiro, M. Hermenegildo, Teaching Prolog with Active Logic Documents, in: D. S. Warren, V. Dahl, T. Eiter, M. V. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog - The Next 50 Years, number 13900 in LNCS, Springer, 2023, pp. 171–183. URL: https://cliplab.org/papers/ActiveLogicDocuments-PrologBook.pdf.

[4] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, J. F. Bastien, Bringing the web up to speed with webassembly, in: A. Cohen, M. T. Vechev (Eds.), Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017, ACM, 2017, pp. 185–200. URL: https://doi.org/10.1145/3062341.3062363. doi:10.1145/3062341.3062363.

[5] A. Zakai, Emscripten: an LLVM-to-JavaScript compiler, in: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications, SPLASH '11, ACM, New York, NY, USA, 2011, pp. 301–312. URL: http://doi.acm.org/10.1145/2048147.2048224. doi:10.1145/2048147.2048224.

[6] M. Hermenegildo, A Documentation Generator for (C)LP Systems, in: International Conference on Computational Logic, CL2000, number 1861 in LNAI, Springer-Verlag, 2000, pp. 1345–1361.

[7] M. Hermenegildo, J. Morales, The LPdoc Documentation Generator. Ref. Manual (V3.0), Technical Report, UPM, 2011. Available at https://ciao-lang.org.

[8] E. Mera, P. Lopez-Garcia, M. Hermenegildo, Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework, in: 25th Int'l. Conference on Logic Programming (ICLP'09), volume 5649 of *LNCS*, Springer-Verlag, 2009, pp. 281–295.

[9] M. Hermenegildo, J. Morales, P. Lopez-Garcia, M. Carro, Types, modes and so much more – the Prolog way, in: D. S. Warren, V. Dahl, T. Eiter, M. V. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog - The Next 50 Years, number 13900 in LNCS, Springer, 2023, pp. 23–37. URL: https://cliplab.org/papers/AssertionsAndOther-PrologBook.pdf.

[10] M. Hermenegildo, G. Puebla, F. Bueno, P. Lopez-Garcia, Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor), Science of Computer Programming 58 (2005) 115–140. doi:10.1016/j.scico.2005.02.006.

[11] D. Ferreiro, J. Morales, S. Abreu, M. Hermenegildo, 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), volume 385 of *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, Open Publishing Association (OPA), 2023, pp. 324–330. URL: https://cliplab.org/papers/hald-demo-iclp-tc.pdf, see also associated poster at https://cliplab.org/papers/hald-poster-iclp.pdf.

[12] J. Wielemaker, F. Riguzzi, R. A. Kowalski, T. Lager, F. Sadri, M. Calejo, Using SWISH to realize interactive web-based tutorials for logic-based languages, Theory Pract. Log. Program. 19 (2019) 229–261. URL: https://doi.org/10.1017/S1471068418000522. doi:10.1017/S1471068418000522.

[13] P. Flach, K. Sokol, J. Wielemaker, Simply Logical - The First Three Decades, in: D. S. Warren, V. Dahl, T. Eiter, M. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog - The Next 50 Years, number 13900 in LNCS, Springer, 2023.

[14] A. Brecklinghaus, P. Koerner, A Jupyter kernel for Prolog, in: Proc. 36th Workshop on (Constraint) Logic Programming (WLP 2022), Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2022.

[15] G. Sartor, A. Wyner, Teaching Prolog and Logic Programming with Jupyter Notebooks, in:

Proceedings of the 41st ICLP Workshops, 2025.

[16] M. V. Hermenegildo, J. F. Morales, P. Lopez-Garcia, Teaching Pure LP with Prolog and a Fair Search Rule, in: Proceedings of the 40th ICLP Workshops, volume 3799, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3799/paper2PEG2.0.pdf.

[17] R. F. Stärk, The theoretical foundations of LPTP (A logic program theorem prover), J. Log. Program. 36 (1998) 241–269. URL: https://doi.org/10.1016/S0743-1066(97)10013-9. doi:10.1016/S0743-1066(97)10013-9.

[18] M. Hermenegildo, J. Morales, P. Lopez-Garcia, Some Thoughts on How to Teach Prolog, in: D. S. Warren, V. Dahl, T. Eiter, M. V. Hermenegildo, R. Kowalski, F. Rossi (Eds.), Prolog - The Next 50 Years, number 13900 in LNCS, Springer, 2023, pp. 107–123. URL: https://cliplab.org/papers/TeachingProlog-PrologBook.pdf.

Declaration on Generative AI: The authors have not employed any GenAI tools for this paper.