clindocs: A system for automatic documentation of ASP programs

Dominik Gerndt[1], Susana Hahn[1]

[1]Potassco Solutions GmbH, Germany

Abstract

Answer Set Programming (ASP) is widely recognized as a powerful paradigm for knowledge representation and reasoning, yet broadening its adoption beyond the core community remains a challenge. A contributing factor is the historical lack of tooling that developers take for granted in mainstream languages: such as linters, code completion, and documentation generators. This work addresses the documentation gap with *clindocs*, an automated tool that analyzes *clingo* ASP encodings and generates structured, navigable documentation from the source code and lightweight comment annotations. Built as an extension of the popular MkDocs ecosystem, it integrates seamlessly into Markdown-based documentation sites. Its main features include rendered and annotated encoding sections, predicate classification into input and output, dependency graph visualization, and a navigable glossary with per-predicate argument descriptions and file references.

1. Introduction

Answer Set Programming (ASP) [1] has seen growing adoption as its expressiveness and declarative modeling power become increasingly recognized across academic and industrial applications. Yet, reaching beyond the ASP community remains a challenge: developers accustomed to mainstream programming languages expect a mature tooling ecosystem: linting, code completion, testing frameworks, and documentation generators, that ASP has historically lacked [2]. Previous attempts to bridge this gap have proposed integrated development environments such as ASPIDE [3] and SeaLion [4]. These tools, however, focus on providing an integrated environment, driving users away from their preferred editors and workflows, and do not provide a dedicated documentation generator.

ASP encodings tend to be compact and expressive, but as they grow and predicates multiply, keeping track of their purpose, their relationships, and the overall information flow through a program becomes non-trivial even for experienced practitioners. Automatic documentation has been addressed for logic programming before: PlDoc [5] generates predicate documentation for SWI-Prolog from structured comments, but targets only Prolog and produces a standalone web interface rather than integrating into a general-purpose documentation ecosystem. Another attempt to document ASP is the *ASPDoc* system [6], a standalone CLI tool to generate static HTML pages from annotated ASP encodings. To this end, the comment-based annotation language called *LANA (Language for ANnotating Answer-set programs)* [6] is used to define explicit program blocks, atoms, types, conditions, and other elements of the encoding. However, the documentation generated by *ASPDoc*, being a standalone system, is not easily integrated into a larger documentation ecosystem, which may limit its usefulness for projects that require comprehensive documentation beyond ASP encodings. Furthermore, the LANA annotation language, while powerful, is very verbose and requires a significant learning curve, which may discourage users from adopting it for their projects.

In contrast, *clindocs* follows the same spirit while targeting ASP encodings with simpler, more intuitive annotations that lower the adoption barrier and embedding seamlessly into a modern and widely adopted documentation ecosystem. It takes as input an encoding and annotations in the form of comments, producing structured, navigable documentation directly from the source code; following the same *doc-as-code* philosophy that has proven successful in other languages. We target encodings

Joint Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026), co-located with the Federated Logic Conference (FLoC 2026), July 18–19 and 24–25, 2026, Lisbon, Portugal.

dominik.gerndt@potassco.com (D. Gerndt); susana.hahn@potassco.com (S. Hahn)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-03

CEUR-WS.org/Vol-4311/EULP_short_03.pdf

written in *clingo* [7], one of the most widely used ASP systems, and build on a dedicated Tree-sitter grammar that covers the full *clingo* syntax including documentation annotations. The *clindocs* system is built as a handler for the `mkdocstrings` plugin, integrating natively into MkDocs [1] and the mkdocs-material theme [2], which are widely adopted tools that generate modern, searchable documentation sites from Markdown sources. This design choice means that ASP projects can adopt *clindocs* with minimal overhead and combine it freely with hand-written prose, diagrams, and examples in a single documentation site.

The main features of *clindocs* are: rendering annotated encodings with inline Markdown prose; analyzing predicates to classify them as input or output; generating a navigable glossary of predicate documentation from structured block comments; producing a compact summary table of all predicates; and visualizing inter-predicate dependencies as a graph. This paper presents the system and its features through a running Sudoku example, illustrating how *clindocs* can turn a bare ASP encoding into a fully documented, browsable reference.

2. The *clindocs* System

The system is openly available at `https://potassco.org/clindocs`. It was developed under the Potassco[3] umbrella, which hosts a collection of tools and libraries for ASP development. Below we describe in detail how it integrates with the MkDocs ecosystem and the documentation features it provides.

2.1. MkDocs Integration

MkDocs is a static site generator oriented towards project documentation, producing clean HTML sites from Markdown source files. The mkdocs-material theme extends it with a polished visual design, rich navigation, and a plugin/extension ecosystem that is widely adopted in open-source projects. *clindocs* is implemented as a *handler* for the `mkdocstrings` plugin [4], which provides a generic mechanism for injecting auto-generated documentation into MkDocs pages. By registering the handler, *clindocs* intercepts directives in Markdown files and replaces them with rendered ASP documentation at build time. Throughout this section we use the Sudoku encoding found in Listing 1 as a running example. To include the documentation of this encoding in a Markdown page, the author adds a directive to any Markdown page using the following syntax:

```
::: sudoku.lp
    handler: asp
    options:
      encodings: true
      glossary: true
      predicate_table: true
      dependency_graph: true
```

The path to the file is resolved relative to the project root. The `handler: asp` key tells *clindocs* to process the file. The `options` block selects and configures which documentation sections are generated. The `options` block supports the following top-level keys, each controlling one generated section: `encodings`, `glossary`, `predicate_table` and `dependency_graph`. Each option is described in detail in the subsections that follow. Apart from the content that is generated from these options, *clindocs* also generates a navigation section (Figure 2) that can be used to quickly jump to the different sections of the documentation, including the list of available predicates.

[1] `https://www.mkdocs.org/`
[2] `https://squidfunk.github.io/mkdocs-material/`
[3] `https://potassco.org`
[4] `https://mkdocstrings.github.io/`

```
%# Sudoku Puzzle
% Uses the well known [puzzle](https://en.wikipedia.org/wiki/Sudoku)

#include "doc.lp".

#const dim = 3.
val(1..dim*dim).
pos(X,Y) :- val(X), val(Y).

%% Internal comment
subgrid(X,Y,(((X-1)/dim)*dim+((Y-1)/dim))) :- pos(X,Y).

% A choice rule for each cell
1 { sudoku(X,Y,V) : val(V) } 1 :- pos(X,Y).

%## Constraints
:- sudoku(X,Y,V), sudoku(X',Y,V), X != X'.
:- sudoku(X,Y,V), sudoku(X,Y',V), Y != Y'.
:- sudoku(X,Y,V), sudoku(X',Y',V), subgrid(X,Y,S), subgrid(X',Y',S), (X,Y)!=(X',Y').

sudoku(X,Y,V) :- initial(X,Y,V).

%## Shows
#show .
#show sudoku/3.
```

Listing 1: Sudoku encoding with Markdown comments (`sudoku.lp`).

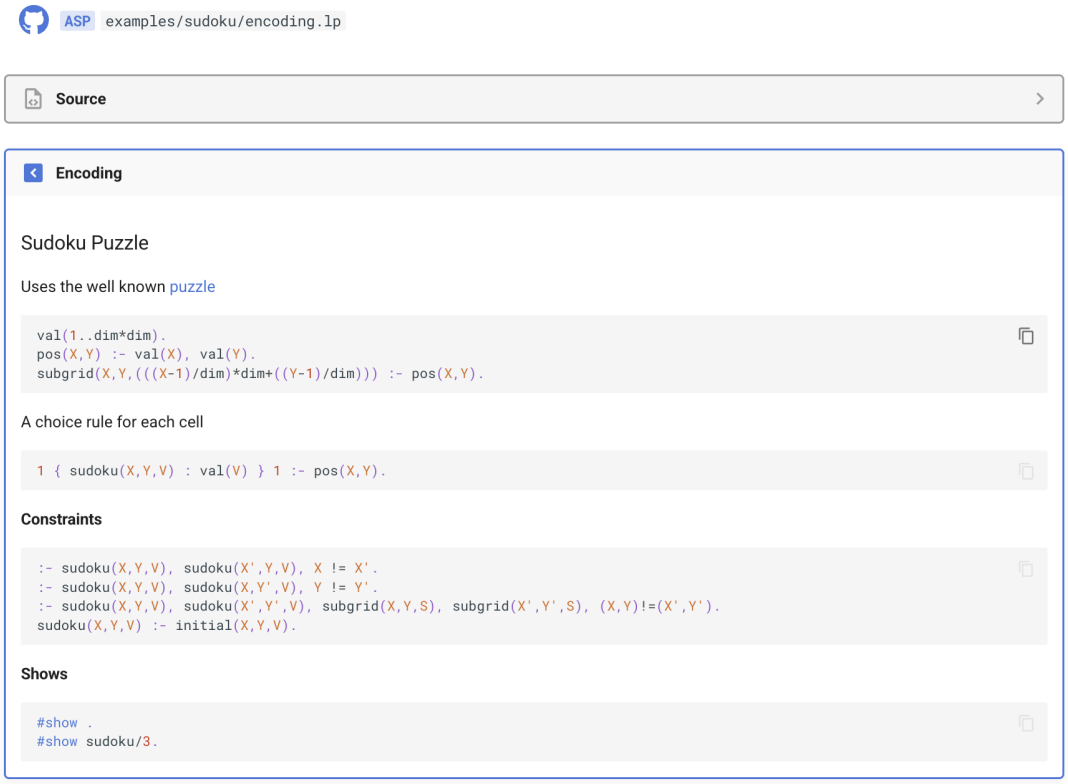


Figure 1: Rendered encoding section for the Sudoku example.

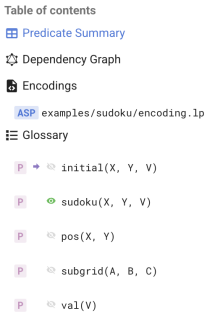


Figure 2: Navigation menu generated for the Sudoku encoding.

2.2. Encoding Rendering

The `encodings` option instructs *clindocs* to render the ASP source file as a readable, annotated document. Inline comments written in Markdown are extracted and typeset as prose, while clingo rules and facts are preserved as formatted code blocks. This allows authors to write literate-style documentation directly inside the encoding. Single-`%` comments are rendered as Markdown prose, while those starting with `%%` are treated as non-documentation comments and ignored in the output. Two sub-options extend the section header for each file: `source: true` appends the full, unmodified source as a collapsible code block, giving readers direct access to the raw encoding; `git_link: true` adds a hyperlink to the file in the configured repository, useful for open-source projects that want to point readers to the canonical version-controlled source. For the sudoku example in Listing 1, the rendered documentation is shown in Figure 1. The first comment is rendered in Markdown, as a header followed by the text with the link to the Wikipedia page. The comment starting with `%%` is ignored and not present in Figure 1.

2.3. Predicate Analysis and Documentation

At build time, *clindocs* parses the files (including transitively included files via `#include`) and performs a lightweight static analysis of the predicate landscape. Under the hood, this parsing is powered by *tree-sitter-clingo* [5], a robust Tree-sitter grammar that covers the entire *clingo* language, including dedicated support for the documentation strings presented in this paper. For each predicate symbol it determines: the files in which it appears, the rules in which it occurs, and its *type*: an *input* predicate is one that appears in rule bodies or as a conditional but is not defined by any rule head in the encoding; an *output* predicate is one that appears in an explicit `#show` directive. These types drive the visual styling used across all generated sections.

Authors can supply human-readable documentation for any predicate by placing a block comment with the special opening delimiter `%*!` anywhere in the included files. Because this syntax is natively supported by the underlying Tree-sitter grammar, it acts as a standardized documentation format, which will integrate smoothly with upcoming ecosystem tools like *clinsight*, a language server for ASP providing real-time editor support such as hover information, go-to-definition, and code completion. This shared foundation means the same predicate documentation will power both the generated website for the documentation and the editor experience, without any additional effort from the developer.

The comments for predicate documentation may appear in the same file as the rules or in a separate `.lp` file included via `#include`, allowing documentation to be maintained independently of the encoding logic. For our running example, the predicate docstrings are placed in `doc.lp` (Listing 2), which is referenced in Listing 1. The full content of the documentation block is rendered as Markdown, so authors may use any mkdocs-material feature (admonitions, tables, math, etc.). Additionally, special syntax is supported for argument documentation: starting with `Args:` followed by a Markdown list of `name: description` entries which is rendered as a table in the output.

[5] https://github.com/potassco/tree-sitter-clingo

```
%*! sudoku(X,Y,V)
Describes a sudoku board. The value of cell (X,Y) is V.

Args:
    - X: the row of the cell
    - Y: the column of the cell
    - V: the value of the cell
*%
%*! initial(X,Y,V)
Describes initial values of the sudoku board. The value of cell (X,Y) is V.

Args:
    - X: the row of the cell
    - Y: the column of the cell
    - V: the value of the cell
*%
%*! pos(X,Y)
Describes a cell (X,Y) in the sudoku board.

Args:
    - X: the row of the cell
    - Y: the column of the cell
*%
%*! val(V)
Describes a possible value V for a cell.

Args:
    - V: the value
*%
```

Listing 2: Sudoku predicate documentation (`doc.lp`).

2.4. Dependency Graph

The `dependency_graph` option generates a graph showing how predicates depend on one another across the included encodings. An edge is drawn from predicates in the body of a rule to those in the head, if the predicate appears negated in the body the edge is shown in a dotted style. For our example, the graph generated for the Sudoku encoding is shown in Figure 3. Input predicates are rendered in purple and shown (output) predicates in green, giving an immediate visual overview of the information flow through the encoding.

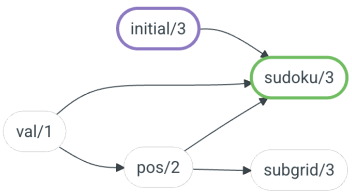


Figure 3: Dependency graph generated for the Sudoku encoding.

2.5. Predicate Glossary

The `glossary` option generates a reference section, with one entry per predicate. Each entry renders the predicate's docstring as formatted Markdown and lists the files in which the predicate is used. Every

predicate also receives a navigation entry in the page's table of contents, making large encodings easy to browse. The section supports two sub-options: `include_references` controls whether the per-file usage list is shown; `include_navigation` controls whether TOC entries are generated. Predicates without docstrings can be excluded globally via the `include_undocumented` flag. Figure 4 shows the glossary section generated for predicate `initial/3` from the comment docstring in Listing 2. The section provides the summary of the predicate, a table with the arguments and their description, and a list of the rules in which the predicate appears.

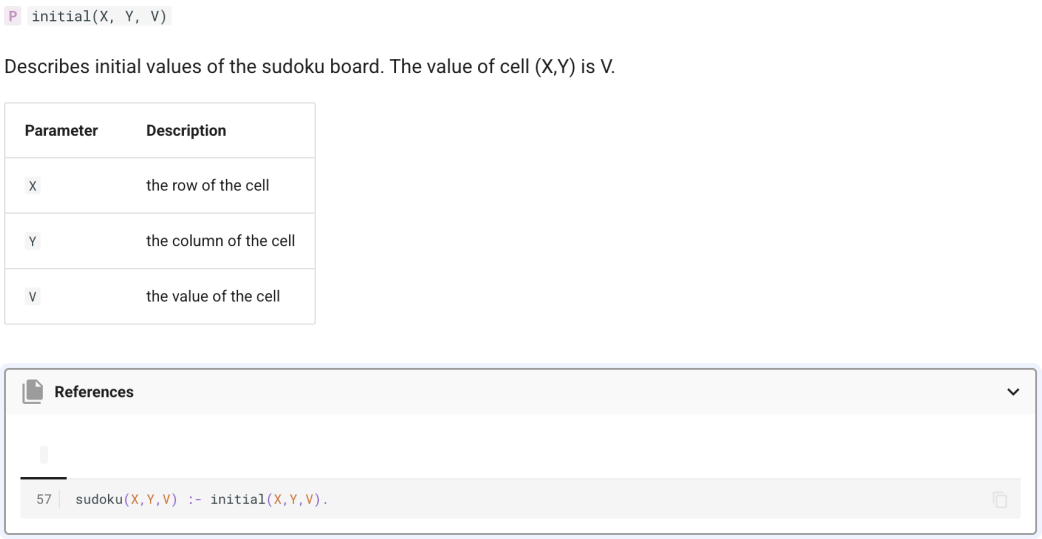


Figure 4: Glossary section showing predicate `initial/3` with usage references.

2.6. Predicate Summary Table

The `predicate_table` option generates a compact summary table listing all predicates in the included encodings together with a short description extracted from the docstring and a type indicator. It provides readers with a quick overview before they dive into the full glossary or the encoding text. The type column uses four symbols: *Input* (indicated by a purple arrow) for predicates not defined by any rule head; *Shown explicitly* (indicated by a green eye icon) for predicates appearing in a `#show predicate/arity` directive; *Shown partially* (indicated by a green eye icon with a plus) for predicates shown via a conditional `#show` statement; and *Hidden* (indicated by an opaque crossed out eye icon) for predicates that are defined but not shown. Figure 5 shows the predicate summary table generated for the Sudoku encoding. The `initial/3` predicate is classified as an input predicate since it does not appear in any rule head, while `sudoku/3` is the only one shown explicitly, classifying it into an output predicate.

3. Conclusion

The quest to extend ASP with tools that simplify development is an ongoing one. With *clindocs* we have taken a step towards a more integrated ecosystem for ASP, providing a tool that automatically generates structured documentation from annotated encodings. We hope this system and its shared foundation with upcoming tools encourage the community to document and maintain their encodings. In the future, we plan to investigate an LLM-assisted documentation workflow where the system suggests documentation content based on the encoding structure, as well as tighter integration with linting and other tools to further close the gap with mainstream language ecosystems.

Name	Definition	Type
initial/3	Describes initial values of the sudoku board. The value of cell (X,Y) is V.	
sudoku/3	Describes a sudoku board. The value of cell (X,Y) is V.	
pos/2	Describes a cell (X,Y) in the sudoku board.	
subgrid/3		
val/1	Describes a possible value V for a cell.	

Figure 5: Predicate summary table for the Sudoku encoding.

Declaration on Generative AI

Generative AI was used exclusively for grammatical and language editing of this manuscript. All content was reviewed and edited by the authors, who take full responsibility for it. All ideas and scientific content are the authors' own.

References

[1] V. Lifschitz, Answer Set Programming, Springer-Verlag, 2019. doi:10.1007/978-3-030-24658-7.

[2] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[3] O. Febbraro, K. Reale, F. Ricca, ASPIDE: Integrated development environment for answer set programming, in: J. Delgrande, W. Faber (Eds.), Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11), volume 6645 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2011, pp. 317–330. doi:10.1007/978-3-642-20895-9_37.

[4] P. Busoniu, J. Oetsch, J. Pührer, P. Skočovskỳ, H. Tompits, Sealion: An eclipse-based ide for answer-set programming with advanced debugging support, Theory and Practice of Logic Programming 13 (2013) 657–673.

[5] J. Wielemaker, A. Anjewierden, Pldoc: Wiki style literate programming for prolog, arXiv preprint arXiv:0711.0618 (2007).

[6] M. D. Vos, D. G. Kisa, J. Oetsch, J. Pührer, H. Tompits, LANA: A language for annotating answer-set programs, In Proceedings of the 14th International Workshop on Non-Monotonic Reasoning (NMR 2012), Rome, Italy, 2012. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-63960.

[7] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, S. Thiele, Potassco User Guide, 2 ed., University of Potsdam, 2015. URL: http://potassco.org.