Possible Worlds Explorer: Datalog and Answer Set Programming for the Rest of Us

Sahil Gupta[1], Yi-Yun Cheng[2], and Bertram Ludäscher[1,2]

[1] Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
`sjgupta2@illinois.edu`

[2] School of Information Sciences, University of Illinois at Urbana-Champaign, USA
`{yiyunc,ludaesch}@illinois.edu`

Abstract. Datalog and Answer Set Programming (ASP) are powerful languages for rule-based querying and constraint solving, respectively. We have developed *Possible Worlds Explorer* (PWE), an open source Python-based toolkit that employs Jupyter notebooks to make working with Datalog and ASP systems easier and more productive. PWE can parse output from different reasoners (Clingo and DLV) and then run analytical queries over all answer sets or "possible worlds" (PWs), e.g., to calculate relative frequencies of atoms across PWs or to hierarchically cluster PWs based on user-defined complexity and similarity measures. PWE also has support for *well-founded Datalog* models (from DLV) and *temporal* models that use a special state argument. Using simple Python functions, generic as well as user-definable presentation and visualization formats can be easily created, e.g., to display all PWs (*world views*), the unique three-valued well-founded model (*partial views*), and temporal models (*timelines* and *time series*). We provide containerized versions of PWE that can be run in the cloud or locally. We hope that in this way Datalog and ASP can be made more accessible for a wider audience.

1 Introduction

Datalog has a long and rich history in database foundations and theory [1,15,22], and has seen a recent resurgence in academia and industry [14,16,19,25,28].[3] *Answer Set Programming* (ASP) shares some common roots with Datalog and evolved from the *stable model semantics* [13] of logic programs with non-stratifed negation and disjunction in rule heads. The availability of ASP solvers such as DLV [23,2] and Clingo [12,11], among others, has facilitated new extensions and applications in KR and ML; see, e.g., [9,8] and the various surveys on theory and practice of ASP [10,5,12,24,7].

Despite the significant interest and considerable capabilities for advanced applications, declarative querying and problem solving with Datalog and ASP have not found wider adoption among general programmers and data and information scientists, and all too often remain an "experts only" domain.

[3] See also the "Datalog 2.0" workshop series [26,3], documenting this resurgence.

CEUR-WS.org/Vol-2368/paper5.pdf

Python, on the other hand, is one of the most popular and fastest growing programming languages in recent years [29]. Some of the reasons for this include its gentle learning curve (from beginner to expert), the comprehensive package support for data science and machine learning (e.g., Pandas, Matplotlib, Scikit-Learn, etc.), and a very active, ever-growing community of users. These and other factors have made Python the de-facto programming language for data science and tool integration.

We have developed *Possible Worlds Explorer* (PWE) [17], an open source, Python-based toolkit that aims to bring together the best of both worlds in order to serve a wide audience of users. In particular, the goals of PWE are to:

(i) empower traditional user groups, i.e., not yet experienced with declarative problem solving, to explore problems and their Datalog/ASP solutions in a familiar, interactive user environment (Jupyter notebooks);
(ii) empower Datalog and ASP experts to be more productive by being able to seamlessly mix and match declarative rules from different rule engines and solvers (e.g., Clingo and DLV) with procedural code (implementing, e.g., analytical queries over and visualizations of possible worlds); and
(iii) allow both groups of users to easily create and share executable knowledge artifacts as Jupyter notebooks [21].

We continue to improve and extend the functionality of the PWE toolkit [17], along with a growing repository of introductory PWE notebooks [18]. To further lower the barrier to entry, PWE and notebooks can be deployed in cloud-based environments, i.e., Whole Tale [6] and Binder [20], so that users can execute the notebooks without having to install any software and dependent packages.

The remainder of the paper is organized as follows: In Section 2, we describe the PWE architecture and discuss its components. In Section 3, we present a few introductory examples for declarative problem solving with PWE. These are meant to provide a first illustration of the power of logic-based declarative problem solving[4] for novices, and to introduce some capabilities of Python-based notebooks (e.g., visualizations of PWs) to Datalog/ASP experts not familiar with Jupyter notebooks. In Section 4, we briefly summarize and conclude.

2 Possible Worlds Explorer: System Overview

PWE is a Python-based open-source toolkit with several components that aim to make interacting with ASP and Datalog systems easier, and to provide users with features and conventions that allow them to analyze potentially large sets of possible worlds (*answer sets*). We employ Jupyter notebooks[5] as the preferred user interface for PWE, since notebooks can interleave an explanatory narrative with code snippets and inline visualizations. In this way, PWE notebooks offer both an interactive problem solving and exploration environment, and can

[4] many more can be found in [10,5,12,24]

[5] https://jupyter.org/

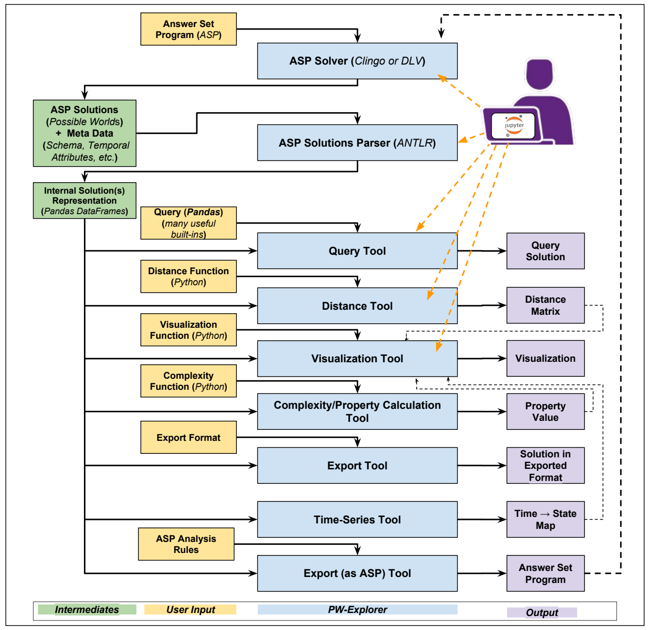


Fig. 1. Possible Worlds Explorer (PWE): an ASP solver (e.g., Clingo, DLV) generates answer sets (PWs). PWE can then query across all PWs and visualize, rank, and cluster them based on user-defined distance and complexity functions. A Jupyter notebook deployment of PWE provides a user-friendly, interactive user interface.

be seen as reproducible, computational artifacts that are shareable, browsable, and re-executable [18,6]. The PWE components can be accessed in conventional scripts and interactive notebooks, like any other Python library. We provide a comprehensive set of Python modules with many functions as part of our PWE library. We also provide a command-line interface for users who prefer to use the command-line. The following are some key components of PWE:

ASP/Datalog Wrappers. PWE currently includes wrappers for Clingo [12,11] and DLV [23,2]. The PWE wrappers provide users with a "pythonic" interface to these engines. Python strings can be used as ASP rules to be evaluated by the underlying reasoners. This also allows users to easily generate rule instances based on a *parameterized* problem. For example, the well-known *Towers of Hanoi* problem is parameterized by the number of disks and pegs. PWE uses ANTLR[6]-based parsers to read the outputs of the wrapped systems and load them into PANDAS[7] DataFrame objects. Another PWE feature is the support for 3-valued, well-founded models [30] output by DLV – see Section 3.2 for an example. Our system also contains extension points so users can add special annotations as comments in logic programs, which then can be detected by PWE. Attribute names, e.g., can help users better understand complex relational schemas, so

[6] **An**other **T**ool for **L**anguage **R**ecognition: https://en.wikipedia.org/wiki/ANTLR
[7] Python Data Analysis Library: https://pandas.pydata.org/

users can include schema annotations of the form

$$\% \texttt{ schema } R(A_1, \ldots, A_n)$$

indicating that relation R has attributes $A_1, \ldots, A_n$. These user-declared names for attributes A_i can then be used as column names in PANDAS DataFrames.

Similarly, the PWE annotation

$$\% \texttt{ temporal } R(\ldots, \texttt{T}, \ldots)$$

identifies, via the special symbol "`T`", the argument position in a relation R that should be treated as containing a temporal or state identifier. This argument position can then be used by other PWE modules for time-series based analysis.

Query Tool. PWE can query across all possible worlds, e.g., to find unique features, or to intersect or union all PWs. In particular, PWE can access and work with *brave* and *cautious* consequences of ASP programs.

Distance Tool. Often a natural question we want to answer when analyzing sets of PWs is: How similar or different are PWs to each other? Using this tool, users can find the distance between any two PWs based on their shared (or unique) facts, or by defining a new distance metric.

Visualization Tool. This module allows users to visualize individual solutions or the whole solution space. Some built-in visualizations, including *clustering*, *dendrograms*, and *2D distance mapping*, all use the distance matrix generated by the distance tool. Users can also easily define their own visualization functions. PWE also includes some basic visualizations for time-series based answer sets, so users can see how states "evolve" over time within each PW.

Complexity (World Feature) Calculation Tool. This tool allows users to define a metric to be calculated for each PW, which then can be used to rank PWs. For example, in the *stable marriage problem* [8], a fairness metric can be defined to find the PW which is fairest for both groups.

Time-Series Module. PWE has some built-in support for ASP problems that are time-series based, e.g., the classic Towers of Hanoi puzzle. The `temporal` annotation mentioned above allows users to indicate which argument holds the state identifier. PWE can then group and display answers by state, in order, etc.

PW Import/Export Tool. This tool is for advanced users who would like to (i) export PANDAS DataFrames used by PWE to other formats (e.g., CSV, Pickle, or SQLite) for further processing, or (ii) re-import as database facts the possible worlds output in an earlier step. PWE can also export a unique 3-valued well-founded model (from DLV) and then re-import it as database facts for a subsequent Clingo step. In addition, different export schemas are available, e.g., "as-is", or a triple-based generic encoding that reifies relation names as data, thereby supporting querying and reasoning about schemas.

3 Declarative Problem Solving by Example

In order to demonstrate the capabilities and features of PWE, we have created several executable Jupyter notebooks which guide the user, step by step, through a number of introductory and advanced problems. In this section we highlight a few elements of some of these examples from our growing demo repository [18]. The repository includes an `environment.yml` file and a `postBuild` bash script that are used to create an automated build to assemble an image to execute PWE and the example notebooks, either locally using Conda[8], or online in a cloud-based environment such as provided by Binder [20] or Whole Tale [6].

3.1 Combinatorics (Graph Coloring): Two ASP Rules Suffice

To get a first intuition of declarative problem solving, consider the following:

```
(a) node(1..5). e(1,2). e(2,3). e(3,4). e(4,1). e(4,5). % Database facts
(b) col(X,red) ; col(X,green) ; col(X,blue) :- node(X). % Generator
(c) :- e(X,Y), col(X,C), col(Y,C), X != Y.              % Constraint
```

Line (a) states that we have a graph consisting of five nodes and some edges $\mathtt{e}(x, y)$, forming a "square" 1-2-3-4-1 with an "appendage" 4-5. Thus line (a) consists of a set of database *facts*. In contrast, a *rule* like the one in (b) is written in the form "*head* `:-` *body*" and means that if the condition in *body* is true, then the *head* logically follows. Here, (b) states that every node x can have any of three colors, i.e., the semicolons ";" in the head are interpreted as logical disjunctions "$\vee$" and (b) thus acts as a *generator* of alternatives (x could be `red` or `green` or `blue`). Finally, (c) is a special rule that acts as a *denial constraint*: it has an *empty* head and states that if there is an edge $\mathtt{e}(x, y)$ and if x and y have the same color c (for $x \neq y$), then the "current world" under consideration is *not* a possible world, since neighboring nodes must always have different colors.

There are $3^5 = 243$ different ways overall to associate one of three colors to five nodes. If the denial constraint (c) is removed, an ASP reasoner will indeed generate all 243 possible worlds. For a human, it is difficult to determine how many and which of the 243 candidate solutions also satisfy the given constraint (c), demanding different colors for neighboring nodes. ASP solvers such as DLV or Clingo easily handle such combinatorial puzzles:

```
$ clingo -n0 3col.lp4
Answer: 1
col(1,blue) col(2,red) col(3,green) col(4,red) col(5,blue)
...
Answer: 36
col(1,red) col(2,green) col(3,blue) col(4,green) col(5,blue)
SATISFIABLE

Models       : 36
CPU Time     : 0.001s
```

[8] https://conda.io/

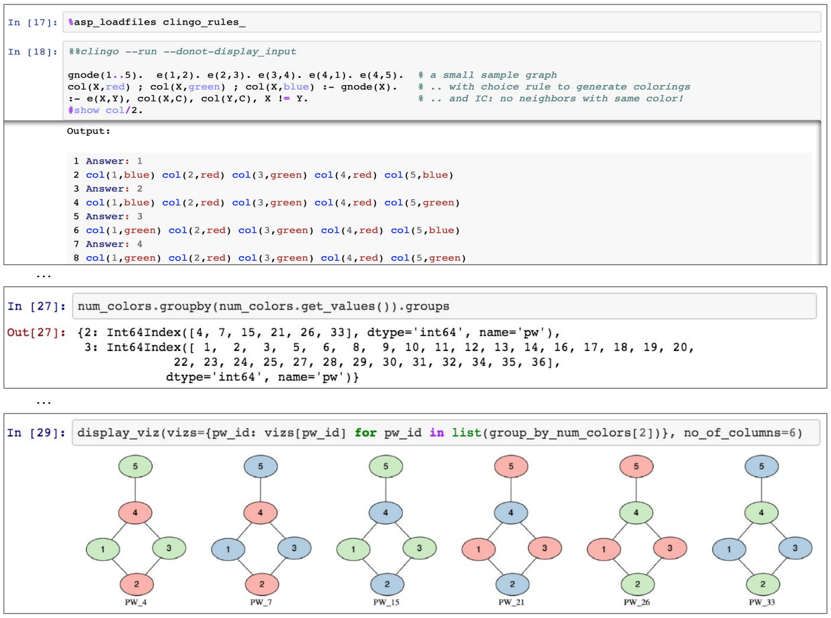


Fig. 2. Screenshots from a Jupyter notebook running PWE: the *line magic* in `[17]` loads ASP rules into cell `[18]` where they can then be edited and run. In `[27]` a `groupby` operation is used to identify the 6 PWs that use only two colors and the 30 PWs that use all three. The visualization in `[29]` shows that PWs in the first group all exhibit the same "color pattern", i.e., PWs are the same modulo renaming of colors. PWE rules can also be used to cluster PWs and find the equivalence classes of all patterns.

The command-line parameter $-\mathtt{n}N$ indicates that at most N models should be computed; setting $N{=}0$ as above means: compute all solutions. Here there are exactly 36 solutions among the 243 candidates, representing all valid 3-colorings of the input graph.

Even for such a fairly small number of PWs, it is difficult for a user to relate and compare solutions and answer simple questions such as "how many PWs use only two colors?" or "what PWs use all three colors?", etc.

With the analysis and visualization capabilities of PWE running within a Jupyter notebook, the limitations of traditional, "spartan" command-line interfaces can be easily overcome: Fig. 2 shows different cells of a demo notebook, highlighting how "meta-analyses" can be performed across all solutions in PWE.

The visualizations in Fig. 3 illustrate how PWE can be used to find further structure in a set of possible worlds. The heat map in the lower left clearly shows the separation of the cluster of 6 solutions using only two colors (cf. Fig. 2) and a larger cluster of the remaining 30 solutions which use all three colors. The latter contains further substructure (revealed by another metric) as seen by the heatmap and associated dendrograms on the right in Fig. 3; see [18] for details.

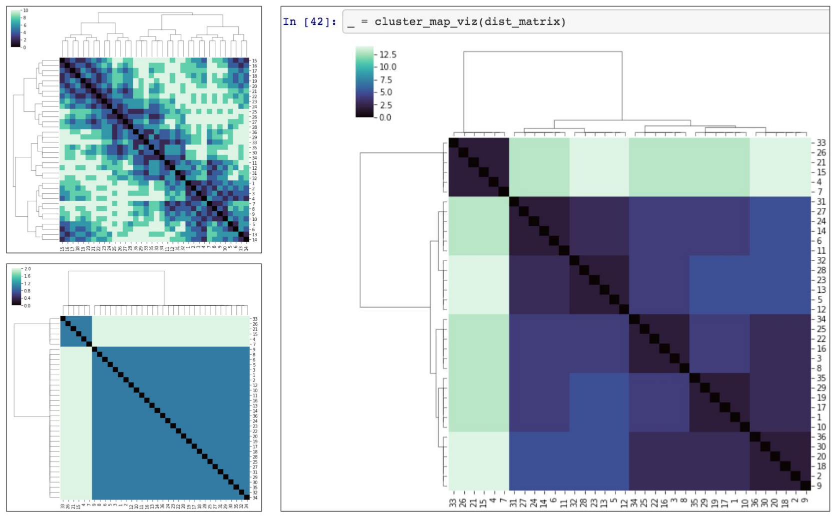


Fig. 3. Different complexity measures and distance metrics when clustering the 36 PWs of valid 3-colorings: An unsuitable metric counts coloring differences directly and "overfits" (upper left); a simple metric obtained by counting how many colors were used works very well and identifies 6 bicolored PWs and 30 tricolored ones (lower left); the cluster of 30 has further substructure corresponding to "coloring patterns" (right).

3.2 Games and 3-Valued (Partial) Models

Consider the single rule Datalog program with recursion through negation:

```
win(X) :- move(X,Y), not win(Y).
```

This (non-stratified) rule states that in a game graph the position x is won, if in the graph there exists a move from x to some position y which is not won (since the opponent then moves from y). Consider a simple graph consisting of the following relation $\texttt{moves} = \{(\texttt{a},\texttt{b}), (\texttt{b},\texttt{a}), (\texttt{b},\texttt{c}), (\texttt{c},\texttt{d})\}$. Since `d` is a sink, `win(d)` is false, i.e., `d` is lost. This in turn makes `win(c)` true since there is a move from `c` to `d`. On the other hand, `a` and `b` are both *drawn*, since the best move from `b` is to `a`, perpetuating the game (moving to `c` would be a win for the opponent). The unique 3-valued *well-founded model* [30] can be computed with DLV directly:

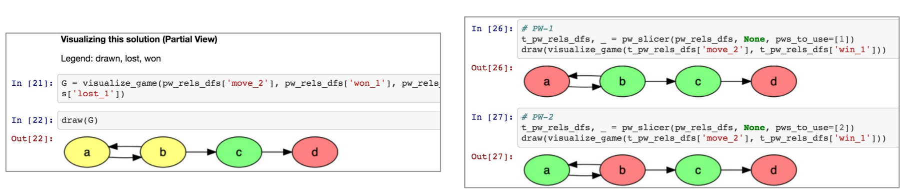
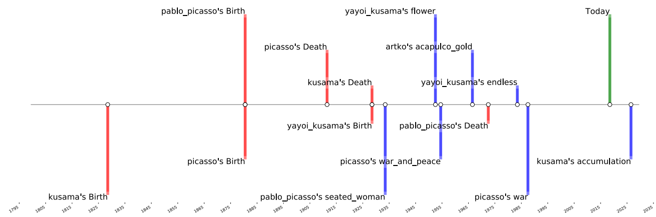


Fig. 4. A partial model $\mathcal{W}$ with *won*, *lost*, and *drawn* positions (left), computed by DLV. The total (i.e., 2-valued stable) models $\mathcal{S}_i$ are computed by DLV or Clingo (right).



Timeline Input Database ________________________________.

Fig. 5. A timeline visualization in PWE, showing elements of an inconsistent database. Prior to any repair actions for MoMA, there is single PW containing all inconsistencies: e.g., there is more than one Picasso with different death years, some artwork was painted posthumously, etc.

```
$ dlv -wf win.dlv
True: {move(a,b), move(b,a), move(b,c), move(c,d), win(c)}
Undefined: {win(b), win(a)}
```

PWE recognizes 3-valued output by DLV. A user-defined visualization is used to render the solved game graph with colored positions (Fig. 4). Both Clingo and DLV can be used to generate all stable models of the above program. All stable models $\mathcal{S}_i$ extend the unique well-founded model $\mathcal{W}$, i.e., agree on the defined parts of $\mathcal{W}$ and can only vary over the undefined parts of $\mathcal{W}$.

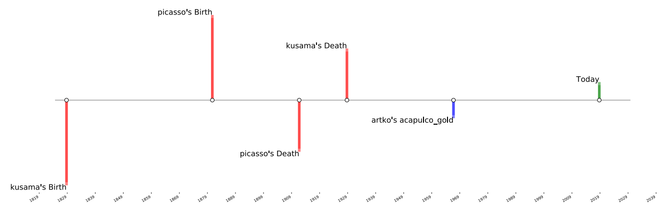
3.3 Integrity Constraint Repair and Timeline Visualizations

ASP has numerous applications, e.g., in model-based diagnosis and repair. In the case of inconsistent databases, alternative *repairs*, i.e., minimal changes that restore integrity, can be generated via declarative rules [4], giving rise to different possible worlds. We illustrate how this can be done with PWE using a small sample dataset from the Museum of Modern Art (MoMA) that we have prepared so it exhibits interesting data quality problems. We selected a handful of artists and a few of their artworks, i.e., Pablo Picasso, Yayoi Kusama, and Artko. Artworks include Picasso's *Seated Woman* and *War and Peace*, Kusama's *Flower*, *Endless*, and *Accumulations*, and Artko's *Acapulco Gold*.

For each of the artists, we created a few synthetic errors to mimic similar quality issues for more poorly curated collections. For Pablo Picasso, e.g., we created another instance Picasso (i.e., without his first name) who passed away in a different year, but who uses the same artist-ID. Integrity constraint violations (ICVs) can be captured via auxiliary rules: e.g., the following rules report primary key violations and "posthumous art works", respectively:

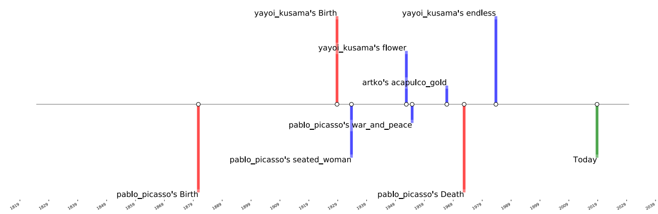
```
icv_PK(ID, N1, N2) :-
  artist(N1,ID,_,_), artist(N2,ID,_,_), N1 != N2.

icv_PostHumous(T, AN, AID, ArtYear) :-
  artwork(T,_,AID,ArtYear),artist(AN,AID,_,DeathYear), ArtYear>DeathYear.
```



Timeline PW_1

⋮



Timeline PW_4

Fig. 6. In PW_1 four artworks are "lost" since the "wrong" artists were deleted. In PW_4, the largest number of artworks (here: 5) are visible, an indication that this PW (and the associated repairs) are preferable over the alternative solutions.

Repair rules then specify alternate "fixes", e.g., by deleting artist `N1` or `N2`, referred to in the rule above. Similar to the way illustrated in Section 3.1, PWE can now be used to rank, compare, and cluster PWs, i.e., alternate "repair worlds". Here, we instead choose a *timeline visualization* (Fig. 5), as it lets users visually inspect the dataset for inconsistencies.

When repair rules are used (see, e.g., [4]), multiple PWs are generated. Fig. 6 shows two of four PWs generated for our running MoMA example. For relatively small numbers of PWs, the user can visually inspect the alternatives. For larger sets, Python code (or declarative meta-programs) can be used to automatically rank and compare worlds.

3.4 Towers of Hanoi: State-based Search and Graph Analysis

Towers of Hanoi (ToH) is a computer science classic to teach about recursion. It is also a good introductory example for declarative problem solving, where the reasoner has to find a sequence of actions to accomplish a goal while observing a number of constraints. The Potassco/Clingo User Guide [11] contains a simple, state-based problem description of ToH.[9] We use a PWE notebook to better analyze the time-series nature of the solutions to this problem [18]. In our PWE demo notebook, we use a small instance of ToH with three disks and the usual three pegs. The smallest number of moves to transfer all disks from the initial state (labeled `aaa`) to the final state (labeled `ccc`) is 7 in this case.

[9] https://github.com/potassco/clingo/tree/master/examples/gringo/toh

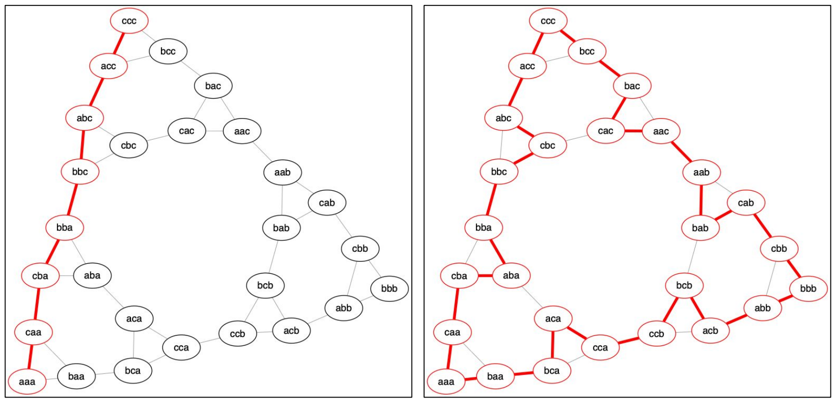


Fig. 7. Hanoi Graph visualizations in PWE: the combination of ASP and powerful graph analytics in Python allows the user to analyze and visualize information encoded in the graph. The shortest path (i.e., the sequence of moves) from the initial state (`aaa`) to the desired state (`ccc`) is shown on the left and takes 7 steps. A Hamiltonian cycle can be computed in the usual way using ASP rules, and then visualized in PWE (right).

Another model of ToH can be created by employing a different representation, the *Hanoi graph*[10]. The state space of this move graph is more interesting than it might seem at first glance. It contains the set of all possible configurations and all valid moves between them and is radially symmetric. By design it also encodes some useful information such as the shortest sequence of moves required to move between any two states. We can create, analyze, and visualize the Hanoi graph easily in PWE: We can use Python code, parametrized by the number of disks and pegs, to generate ASP rules that encode the graph edges. We then leverage Python's NetworkX[11] library to extract and conveniently visualize this information. We can then perform various analyses on the Hanoi graph. For example, as shown on the left in Fig. 7, we can find the shortest sequence of moves that take us from the initial state (`aaa`) to the goal state (`ccc`) by finding the shortest path between these two nodes. Similarly, one might be interested in determining whether there exists a sequence of moves that visits all possible configurations exactly once and returns to the initial configuration. This test for the existence of a *Hamiltonian cycle* can be directly encoded in ASP, or it could be computed using a Python library; see Fig. 7 (right) for a rendering of the Hamiltonian cycle in the Hanoi graph, and [18] for further details.

4 Summary and Conclusions

We have presented *Possible Worlds Explorer* (PWE), a toolkit Datalog and ASP systems with the down-to-earth practical data wrangling and visualization capa-

[10] `https://en.wikipedia.org/wiki/Tower_of_Hanoi#Graphical_representation`
[11] NetworkX Python Library: `http://networkx.github.io`

bilities of Python, running in an interactive, user-friendly Jupyter environment. To illustrate the power and capabilities of PWE, we have developed multiple demonstration notebooks that are available at [18]. In addition, we have provided a Conda-based virtual environment configuration which can be used with Binder [20] to navigate and interact with these examples and create new ones. We believe that PWE will help making declarative problem solving more accessible to practitioners and educators in academia and industry. Conversely, we hope that theoreticians will be tempted to use PWE as a "language lab", where ideas combining multiple tools and systems can be tried out rapidly. The PWE library is available via PyPi[12] and the source code is available on Github [17].

We have not studied the scalability of PWE yet, but have run a few experiments involving millions of facts, spread across tens to hundreds of thousands of PWs, resulting in PWE load times under 15 minutes on a modern laptop. PWE is an ongoing project and the tool is still evolving and being improved. We believe, however, that PWE in its present form already adds significant value for new users, especially when compared to the "bare bones", command-line only tools usually available. In future iterations of PWE, we intend to add additional functionality to this tool such as support for temporal and range queries and new distance measures that can take into account temporal aspects. We also believe that PWE and tools like it may play an increasing role in future data science and AI applications (cf. QASP [27]) by combining declarative problem solving in the style of ASP and Datalog with tool integration through Python.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G., Terracina, G.: The Disjunctive Datalog System DLV. In: de Moor et al. [26]
3. Barceló, P., Pichler, R. (eds.): Datalog in Academia and Industry: 2nd Intl. Workshop, Datalog 2.0, Vienna, Austria, Sept. 11-13. LNCS 7494, Springer (2012)
4. Bertossi, L.: Database Repairing and Consistent Query Answering. Synthesis Lectures on Data Management **3**(5), 1–121 (2011)
5. Brewka, G., Eiter, T., Truszczyński, M.: Answer Set Programming at a Glance. CACM **54**(12), 92–103 (2011)
6. Brinckman, A., Chard, K., Gaffney, N., Hategan, M., Jones, M.B., Kowalik, K., Kulasekaran, S., Ludäscher, B., Mecum, B.D., Nabrzyski, J., Stodden, V., Taylor, I.J., Turk, M.J., Turner, K.: Computing Environments for Reproducibility: Capturing the "Whole Tale". Future Gen. Comp. Systems **94**, 854–867 (2019)
7. Cabalar, P., Pearce, D., Valverde, A.: Answer Set Programming from a Logical Point of View. KI - Künstliche Intelligenz **32**(2), 109–118 (Aug 2018)
8. Clercq, S.D., Schockaert, S., Cock, M.D., Nowe, A.: Solving Stable Matching Problems Using Answer Set Programming. TPLP **16**(3), 247–268 (2016)
9. Eiter, T., Germano, S., Ianni, G., Kaminski, T., Redl, C., Schüller, P., Weinzierl, A.: The DLVHEX System. KI - Künstliche Intelligenz **32**(2), 187–189 (2018)

[12] PWE PyPi: `https://pypi.org/project/PW-explorer`

10. Eiter, T., Ianni, G., Krennwallner, T.: Answer Set Programming: A Primer. In: Reasoning Web. pp. 40–110. LNCS 5689, Springer (2009)
11. Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S., Wanko, P.: Potassco User Guide v2.2.0. Tech. rep., University of Potsdam (Jan 2019)
12. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on AI and Machine Learning **6**(3), 1–238 (2012)
13. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP. vol. 88, pp. 1070–1080 (1988)
14. Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The Lixto Data Extraction Project: Back and Forth Between Theory and Practice. In: Principles of Database Systems (PODS). pp. 1–12 (2004)
15. Grädel, E., Kolaitis, P.G., Libkin, L., Marx, M., Spencer, J., Vardi, M.Y., Venema, Y., Weinstein, S.: Finite Model Theory and Its Applications. Springer (2005)
16. Green, T.J., Aref, M., Karvounarakis, G.: LogicBlox, Platform and Language: A Tutorial. In: Datalog in Academia and Industry, pp. 1–8. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2012)
17. Gupta, S.: Possible Worlds Explorer source code repository (Feb 2019), `https://github.com/idaks/PW-explorer`
18. Gupta, S.: PWE demos (Feb 2019), `https://github.com/idaks/PWE-demos`
19. Hellerstein, J.: The Declarative Imperative: Experiences and Conjectures in Distributed Logic. SIGMOD Rec. **39**(1), 5–19 (Sep 2010)
20. Project Jupyter, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, Kyle Kelley, Gladys Nalvarte, Andrew Osheroff, Pacer, M., Yuvi Panda, Fernando Perez, Benjamin Ragan Kelley, Carol Willing: Binder 2.0 – Reproducible, interactive, sharable environments for science at scale. In: 17th Python in Science Conference. pp. 113 – 120 (2018)
21. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., Jupyter development team: Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows. In: Positioning and Power in Academic Publishing: Players, Agents and Agendas. pp. 87–90. IOS Press (2016)
22. Kolaitis, P.: A Retrospective on Datalog 1.0. In: Barceló and Pichler [3]
23. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL **7**(3), 499–562 (2006)
24. Lifschitz, V.: Achievements in Answer Set Programming. Theory and Practice of Logic Programming **17**(5-6), 961–973 (2017)
25. Mazuran, M., Serra, E., Zaniolo, C.: Extending the Power of Datalog Recursion. The VLDB Journal **22**(4), 471–493 (Aug 2013)
26. de Moor, O., Gottlob, G., Furche, T., Sellers, A. (eds.): Datalog Reloaded: 1st Intl. Workshop, Datalog, Oxford, UK, March 16-19. LNCS 6702, Springer (2011)
27. Nickles, M.: A System for the Use of Answer Set Programming in Reinforcement Learning. In: Logics in AI, pp. 488–491. Springer (2012)
28. Seo, J., Guo, S., Lam, M.S.: SociaLite: An Efficient Graph Query Language Based on Datalog. TKDE **27**(7), 1824–1837 (2015)
29. Tung, L.: Programming Language of the Year? Python Is Standout in Latest Rankings (Jan 2019), ZDNet Magazine
30. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The Well-Founded Semantics for General Logic Programs. J. ACM **38**(3), 619–649 (Jul 1991)