VADACODE: A Logician-friendly IDE for Datalog$^{\pm}$

Luigi Bellomarini
Bank of Italy
luigi.bellomarini@bancaditalia.it

Andrea Gentili
Bank of Italy
andrea.gentili@bancaditalia.it

Davide Magnanimi
Bank of Italy & Politecnico di Milano
davide.magnanimi@bancaditalia.it

Emanuel Sallinger
TU Wien & University of Oxford
sallinger@dbai.tuwien.ac.at

ABSTRACT

Languages, namely, fragments, of the Datalog+/- family are attracting interest in both academia and industry because of their possibility to balance high expressive power and computational complexity. However, understanding the differences among the fragments, mastering them to achieve scalable industrial applications, and communicating their peculiarities to a non-expert audience is challenging for researchers, developers, logicians, and educators.

In this demo, we introduce Vadacode, an IDE for Datalog+/- designed to support a broad category of users. The tool offers advanced features, including fragment detection, syntax highlighting, code completion, error diagnostics, schema inference, debugging support, and AI-assisted coding capabilities. Thanks to our experience in the financial context, our demo will guide the audience in modeling financial Datalog+/- programs, showcasing a seamless and effective coding experience.

PVLDB Reference Format:
Luigi Bellomarini, Andrea Gentili, Davide Magnanimi, and Emanuel Sallinger. VADACODE: A Logician-friendly IDE for Datalog$^{\pm}$. PVLDB, 18(12): 5411 - 5414, 2025.
doi:10.14778/3750601.3750684

PVLDB Artifact Availability:
An accompanying video is available at https://youtu.be/5GrkwAv4o9A. The source code is in the process of being released as open source and will be made available at https://github.com/bancaditalia/vadacode. The browsable documentation is available at https://vadalog.org.

1 INTRODUCTION AND DEMO OVERVIEW

Conceived as a declarative query language [1, 17], *Datalog* has become a reference in deductive systems [16], logic programming [23], and *Knowledge Representation and Reasoning* (KRR). Applications of ontological reasoning [19], ontology-based data access [12], and Knowledge Graphs [20] call for extensions to Datalog, culminating in the Datalog$^{\pm}$ [14] family of languages (or fragments), which supports *existential quantification* for the creation of new "discourse elements" or variable data items, that is, *labeled nulls*.

Existentials, together with other relevant characteristics such as full recursion, a mild form of negation, algebraic and aggregation functions, and utility libraries [9, 19, 24] made Datalog$^{\pm}$ a reference tool for *deductive AI* as witnessed by the rise of dedicated venues [3], systems [4, 5, 11, 15, 21, 22, 25], and applications [3, 7, 10, 18].

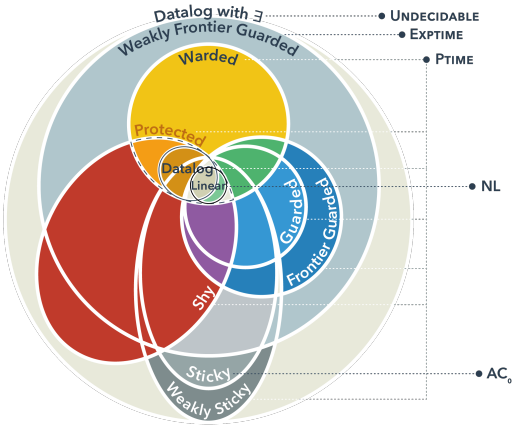

Figure 1: Some relevant fragments of Datalog$^{\pm}$.

Since, in general, Datalog is undecidable in the presence of existential quantification, the restrictions to the interplay between recursion and existential quantification distinguishes the various fragments of Datalog$^{\pm}$. Each of them adopts a different strategy to constrain such interaction and thus strike a balance between expressive power and computational complexity. An overview of the main fragments of Datalog$^{\pm}$ is shown in Figure 1. Some fragments, such as *Linear Datalog*$^{\pm}$ are scalable and simple, others like *Weakly Frontier Guarded* have high expressive power at the cost of exponential time in data complexity. Others like *Warded* are a good tradeoff for practical use, being PTIME in data complexity.

The VADACODE IDE. We intercept the need for a general-purpose Integrated Development Environment (IDE) for Datalog$^{\pm}$, which can serve enterprise developers, researchers, experienced logicians, and educators alike. From developers, we can expect proficiency in translating business requirements into logical languages. However, they are typically unfamiliar with the various fragments—whose recognition and use rely on sometimes intricate syntactic conditions—and require technical support to write fit-for-purpose code. Conversely, researchers and logicians may possess in-depth knowledge of all fragments and seek tools that clearly highlight their syntactic properties while detecting potential violations. Their objectives might include creating research examples, exploring the

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750684

intricacies of these languages, or even developing new ones. Educators teaching databases, logic, KRR, and AI courses would benefit from a didactic tool akin to the didactic compilers used in computer architecture classes. Such a tool would allow students to experiment with these languages, understand their underlying principles, and gain familiarity with query languages and first-order-like logic.

In this demo, we introduce Vadacode, a Datalog$^{\pm}$ IDE that caters to the needs of various user categories while offering a wide range of advanced programming features previously unseen in Datalog-based languages. These include *detection of the most relevant fragments* found in the literature, *syntax highlighting*, *code auto-completion*, *error explanation*, *schema inference*, and *natural language coding*. The IDE has been designed with a focus on the Vadalog language [11], an extended version of *Warded Datalog*$^{\pm}$, used in a variety of applications, particularly in the financial domain, but is deliberately broader than that in terms of supported language fragments (see Fig. 1).

Goal of the Demo. The demo purpose is twofold: first, to showcase and experiment with features that benefit our target user groups, demonstrating that a high-level coding experience is achievable even with seemingly intricate languages originating from the academic community; second, to share our experience in applying Datalog$^{\pm}$ to the development of complex real-world financial tasks, which have successfully led to production systems.

This demo aims to foster the exchange of ideas, patterns, best practices, and use cases from industrial applications. More broadly, we seek to inspire the deductive AI community to bridge the gap between research and industry, highlighting domains where well-defined business logic can be effectively leveraged through declarative approaches. Ultimately, our goal is to take another step toward integrating decades of advancements in logic-based AI with industrial practices, facilitating its transition into mainstream solutions.

Scope of the Demo. Inspired by our work with the Central Bank of Italy, we will guide the audience through the process of encoding financial tasks in Datalog$^{\pm}$. Our demonstration will provide participants with hands-on experience in developing programs within Vadacode, leveraging its suite of features and learning from financial use cases. We will engage both developers and logicians: developers with a rich showcase of Vadacode capabilities, and logicians with an in-depth look at the language features.

The audience will: (i) select a **financial scenario** from a curated set and engage in a guided discussion to achieve a logic modeling; (ii) step into the role of a developer, iteratively **coding a Datalog$^{\pm}$ program** that captures the scenario while exploring Vadacode features along the way. A specific focus will be on **fragment-detection**, demonstrating how developers can leverage it for informed coding, researchers and logicians can analyze its syntactic intricacies, and educators can communicate its concepts to students.

Overview. The remainder of the paper is organized as follows. In Section 2, we lay out, by example, some of the technical background captured by Vadacode. In Section 3, we describe the system. Finally, in Section 4, we illustrate the organization of the demo. An accompanying video is available.[1]

[1] https://youtu.be/5GrkwAv4o9A

2 A DATALOG$^{\pm}$ JOURNEY

Consider a database D of companies (company), their sector of economic activity (activity), and their NACE code (NACE)—the official European classification of such sector of activity. Our Datalog$^{\pm}$ developer seeks to identify all the pairs of competitors, with the following set of rules Σ.

Example 2.1.

$$\mathsf{company}(x) \rightarrow \exists \hat{a}\, \mathsf{activity}(\hat{a}, x) \quad (1)$$

$$\mathsf{activity}(\hat{a}, x), \mathsf{activity}(\hat{a}, y) \rightarrow \mathsf{competitor}(x, y, \hat{a}) \quad (2)$$

$$\mathsf{competitor}(x, z, \hat{a}), \mathsf{competitor}(z, y, \hat{a}) \rightarrow \mathsf{competitor}(x, y, \hat{a}) \quad (3)$$

$$\mathsf{activity}(\hat{a}, x), \mathsf{activity}(\hat{b}, y), \mathsf{NACE}(n, x), \mathsf{NACE}(n, y) \rightarrow a = b \quad (4)$$

Every company is involved in one activity (Rule (1)), which sometimes is provided in D. Companies involved in the same activities are competitors (Rule (2)), which is a transitive relation (Rule (3)). Companies sharing the NACE are involved in the same activity (Rule (4)).■

Can our developer achieve her business goal by using the rules in Σ? In more technical terms, the questions we want to answer are: (i) is she using a suitable fragment of Datalog$^{\pm}$? (ii) if so, what is the expected data complexity of the reasoning task?

First of all, we observe that Rule (1) uses an existential quantifier, which places us in the Datalog$^{\pm}$ space, confirming that Σ is not in plain Datalog. Next, we observe that Rule (2) joins two atoms, meaning that Σ is not in *Linear Datalog*$^{\pm}$, since such fragment allows only single-atom bodies. In fact, according to some definitions [2], Linear Datalog$^{\pm}$ may admit joins, but only if at most one of the join operands is intensional, i.e., it appears in any rule head. This condition is not met in our example, as activity appears in the head of Rule (1). We can now consider whether Σ belongs to *Guarded Datalog*$^{\pm}$ [13]. For this to hold, every join in Σ must have a "guard", an atom that contains all the variables appearing in the body. However, this requirement is violated by Rule (3), since x appears in the first occurrence of competitor, but not in the second. Therefore, Σ is not even in the guarded fragment.

We are now ready to check whether Σ is in the more expressive, yet still tractable, *Warded Datalog*$^{\pm}$ fragment [19]. This task is more challenging and, towards it, we must introduce some supplementary definitions. Given a predicate p, we call $p[i]$, where $i > 0$ is the i-th term of p, a *position*. Position activity[1] is *affected* since variable $\hat{a}$ in Rule (1) is existentially quantified—and we use the hat ($\hat{a}$) symbol to denote it. Whenever a variable always appears in the body of a rule only in affected positions, then it is *harmful*. It is the case of variable $\hat{a}$ in Rule (2). Harmful variables propagating into the head, called *dangerous*, recursively produce new affected positions, like in the case of competitor[3] in Rule (2). Non-harmful variables are called *harmless*, like x, y, or z in Rule (3). For Σ to be warded, all the dangerous variables of every rule should appear within one single body atom, the "ward", which can join with other atoms only through harmless variables. This condition is violated by Rules (2) and (3), given that the dangerous variable $\hat{a}$ appears in two atoms. Therefore Σ is not warded. Walking up the complexity lines, we can check whether Σ is *Frontier-Guarded* or *Weakly Frontier-Guarded* [6].

For the former, each body must have an atom containing all the variables of the frontier (those shared between the body and the head); for the latter, it should contain at least the affected ones of the frontier. Neither the conditions are fulfilled by Rules (2) and (3). In summary, without even considering Rule (4), we understand that there are no complexity guarantees for Σ.

Our developer then reformulates Σ as a simpler program.

Example 2.2.

$$\mathsf{company}(x) \rightarrow \exists \hat{a}\, \mathsf{activity}(\hat{a}, x) \tag{5}$$

$$\mathsf{activity}(\hat{a}, x), \mathsf{activity}(\hat{a}, y) \rightarrow \mathsf{competitor}(x, y) \tag{6}$$

$$\mathsf{activity}(\hat{a}, x), \mathsf{activity}(\hat{b}, y), \mathsf{NACE}(n, x), \mathsf{NACE}(n, y) \rightarrow a = b \tag{7}$$

Rule (6) here is modified and does not consider the specific activity, which is in fact irrelevant. Rule (3) has been removed, since the transitivity naturally derives from Rule (6). ■

Now, in Rule (6), the variable $\hat{a}$ is not dangerous anymore, but simply harmful. Thus, the absence of a body ward does not prevent wardedness. Therefore, we can preliminarily conclude that Σ in Example 2.2 is warded. Regarding Rule (7), some further considerations are needed. In Datalog$^{\pm}$, we call the rules where the head is an equality as *equality-generating dependencies* (EGDs). In general, the presence of such conditions can alter complexity and even decidability of Σ. In this case, we name the EGD as harmful [8]. Still, our developer can try a final rewriting in plain Datalog, by replacing Rule (7) with $\mathsf{NACE}(n, x), \mathsf{NACE}(n, y) \rightarrow \mathsf{competitor}(x, y)$, and Rule (5) with $\mathsf{competitor}(x, z), \mathsf{competitor}(z, y) \rightarrow \mathsf{competitor}(x, y)$.

We are at the end of our coding experience, which provided at the same time a glimpse on the technical intricacies of Datalog$^{\pm}$ fragments, delving into some background, and the intuition that a specific IDE is of extreme help for many categories of users. Vadacode, among its multiple characteristics, streamlines the process we have discussed in this section.

3 THE SYSTEM

Vadacode originates from our experience in designing complex logic-based reasoning tasks in the economic and financial domains for the Central Bank of Italy. Our goal is to provide internal users with a fast-track path to learning, applying, and enjoying the process of writing logic programs in their respective fields. The team that developed Vadacode followed the practice of dogfooding, using the product to discover new and exciting ways to utilize and enhance it. At its core, Vadacode is a full-featured IDE, developed as an extension of Visual Studio Code (VSCode). Visual Studio Code was chosen for its broad platform support, rich set of features, open-source nature, and the largest user base among IDEs.

System architecture. Vadacode consists of two components of Vadacode: a *client extension* and a *language server*, which implements the open standard Language Server Protocol (LSP). The language server encapsulates all language tooling within its process to avoid impacting editor performance, despite the complexity of the background analysis. A small subset of features (e.g., *projects*) is implemented on the client side.

System features. To support modern development and welcome traditional developers, Vadacode mimics the typical IDE experience by embedding all the features in the mainstream VSCode workflow.
Semantic highlighting. Vadacode offers a form of highlighting that is aware of the internals of the fragments, like those we have described in Section 2. For example, affected positions as well as harmless, harmful, and dangerous variables are pointed out.
Code diagnostics and Fragment detection. Programs are analyzed to identify syntactic and semantic errors, providing fragment-aware suggestions: for example, violations of the fragment limitations are shown in-context with a squiggly line aesthetic (e.g., Figure 2). The tool features full Datalog$^{\pm}$ fragment detection, with additional fragment-related information provided through *hover tooltips*.
Definitions and Code navigation. Even the simple scenario of Section 2 requires the developer to track atoms and variables throughout the program. Vadacode offers source code navigation thanks to an *outline* of used symbols and *atom signatures*, which can then be used to find *references* and *definitions* in one click.
Code documentation. Vadacode fosters good development practices and features code-level documentation of Datalog$^{\pm}$ programs, in the style of Javadoc or Jsdoc. Documentation comments can be added directly to programs. Vadacode scans the program source and generates code hints to enhance the developer's experience.
Other features. Vadacode supports several other features to ease the developer's life, such as: (i) *code actions*, which provide in-context refactoring options, (ii) *atom renaming*, which allows to globally change an atom name, (iii) and *code completion*, which provides suggestions based on semantics such as automatic bindings.

Program and notebook support. Vadacode offers two modes of use. In *program mode*, developers can submit Datalog$^{\pm}$ programs to the reasoner via an HTTP endpoint. The results are then displayed in a dedicated VSCode tab. In *notebook mode*, data scientists can iterate on small programs and interleave them with images, equations, and explanatory text, similarly to Jupyter notebooks.

AI-aided programming. Vadacode integrates with the GitHub Copilot extension to enable natural language interaction with Datalog$^{\pm}$ programs. A dedicated *ChatBot* is included in Vadacode to boost the productivity of experienced developers and accelerate the learning curve for newcomers. Through the chat interface, users can describe code snippets, modify programs, fix issues, scaffold code, and ask a wide range of questions.

4 DEMONSTRATION PLAN

Our presentation script captures the needs of expert and novel users interested in Datalog$^{\pm}$. After a very brief focus on the development setup, the demonstration will start.

Feature tour. First, we introduce Vadacode and give a brief explanation of its architecture, pointing to the online documentation.[2] We will let participants freely explore Vadacode to creatively learn its features through carefully developed educational Datalog$^{\pm}$ programs, designed to stimulate discussion over specific features in real-world settings. Vadacode will support novel users and introduce them to the Datalog family of languages using tool capabilities

[2] https://vadalog.org

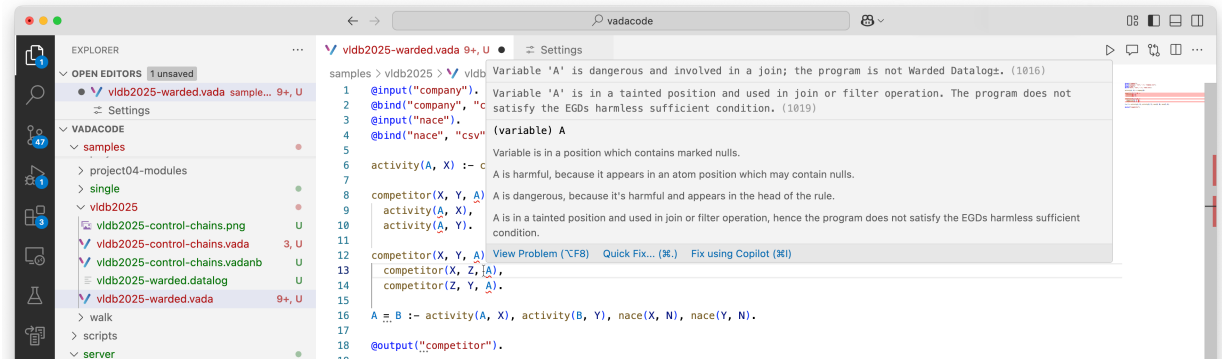

Figure 2: Warded Datalog$^{\pm}$ *fragment detection* in VADACODE at a glance.

like *syntax support* and *code navigation*, along with safe experimentation with thorough *code diagnostics*. Experienced Datalog users will enjoy the support for more advanced logic features such as *fragment detection*.

Corporate knowledge graph scenario. Participants will live the *development experience* by writing a Datalog$^{\pm}$ program supporting the ownership knowledge graph of the Bank of Italy. Without leaving VADACODE, participants will explore the input database and learn how to perform *data ingestion* and *schema inference*. They will then play the role of a *business analyst*, implementing two problems in the corporate economics realm. Using *fragment snippets*, they will implement a solution that combines *full recursion*, *aggregation*, and *stratified negation*. Technical users will appreciate VADACODE for its ability to help them understand, modify, and debug reasoning programs, while enjoying a state-of-the-art development experience. Business-oriented participants will benefit from seeing, in an interactive way, how real-world scenarios can be modeled using logic statements to derive valuable insights.

Fragment debugging. Participants will act as educators, using VADACODE to teach advanced concepts in the Datalog$^{\pm}$ family through novel *explanation metaphors*. They will analyze complex programs and explore literature examples with the help of features such as *inline diagnostics*, *semantic tagging*, and *hover insights*.

ACKNOWLEDGMENTS

This work was supported by the Vienna Science and Technology Fund [10.47379/VRG18013, 10.47379/NXT22018, 10.47379/ ICT2201] and the Austrian Science Fund [10.55776/COE12].

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.
[2] Foto N. Afrati, Manolis Gergatsoulis, and Francesca Toni. 2003. Linearisability on datalog programs. *Theor. Comput. Sci.* 308, 1-3 (2003), 199–226.
[3] Mario Alviano and Matthias Lanzinger (Eds.). 2024. *Datalog-2.0*. CEUR Workshop Proceedings, Vol. 3801. CEUR-WS.org.
[4] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. 2015. Design and Implementation of the LogicBlox System. In *SIGMOD*. ACM, 1371–1382.
[5] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter. 2015. Graal: A Toolkit for Query Answering with Existential Rules. In *RuleML (LNCS)*, Vol. 9202. Springer, 328–344.
[6] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. 2011. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *IJCAI*. IJCAI/AAAI, 712–717.
[7] Teodoro Baldazzi, Luigi Bellomarini, and Emanuel Sallinger. 2023. Reasoning over Financial Scenarios with the Vadalog System. In *EDBT*. 782–791.
[8] Luigi Bellomarini, Davide Benedetto, Matteo Brandetti, and Emanuel Sallinger. 2022. Exploiting the Power of Equality-generating Dependencies in Ontological Reasoning. *Proc. VLDB Endow.* 15, 13 (2022), 3976–3988.
[9] Luigi Bellomarini, Davide Benedetto, Georg Gottlob, and Emanuel Sallinger. 2022. Vadalog: A modern architecture for automated reasoning with large knowledge graphs. *Inf. Syst.* 105 (2022), 101528.
[10] Luigi Bellomarini, Marco Favorito, Eleonora Laurenza, Markus Nissl, and Emanuel Sallinger. 2024. Towards FATEful Smart Contracts. In *DLT (CEUR)*, Vol. 3791. CEUR-WS.org.
[11] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. 2018. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *VLDB* 11, 9 (2018), 975–987.
[12] Meghyn Bienvenu, Diego Figueira, and Pierre Lafourcade. 2024. Shapley Value Computation in Ontology-Mediated Query Answering. In *KR*.
[13] Marco Calautti, Georg Gottlob, and Andreas Pieris. 2015. Chase Termination for Guarded Existential Rules. In *AMW (CEUR)*, Vol. 1378. CEUR-WS.org.
[14] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris. 2010. Datalog+/-: A Family of Languages for Ontology Querying. In *Datalog (LNCS)*, Vol. 6702. Springer, 351–368.
[15] David Carral, Irina Dragoste, Larry González, Ceriel J. H. Jacobs, Markus Krötzsch, and Jacopo Urbani. 2019. VLog: A Rule Engine for Knowledge Graphs. In *ISWC (2) (LNCS)*, Vol. 11779. Springer, 19–35.
[16] Stefano Ceri, Anna Bernasconi, Alessia Gagliardi, Davide Martinenghi, Luigi Bellomarini, and Davide Magnanimi. 2024. PG-Triggers: Triggers for Property Graphs. In *SIGMOD Conference Companion*. ACM, 373–385.
[17] Stefano Ceri, Georg Gottlob, and Letizia Tanca. 1989. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *KDE* 1, 1 (1989), 146–166.
[18] Georg Gottlob. 2022. Adventures with Datalog: Walking the Thin Line Between Theory and Practice. In *AI*IA (LNCS)*, Vol. 13796. Springer, 489–500.
[19] Georg Gottlob and Andreas Pieris. 2015. Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue. In *IJCAI*. AAAI Press, 2999–3007.
[20] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2020. Knowledge Graphs. *CoRR* abs/2003.02320 (2020).
[21] Alex Ivliev, Lukas Gerlach, Simon Meusel, Jakob Steinberg, and Markus Krötzsch. 2024. Nemo: A Scalable and Versatile Datalog Engine. In *Datalog (CEUR)*, Vol. 3801. CEUR-WS.org, 43–47.
[22] Herbert Jordan, Bernhard Scholz, and Pavle Subotic. 2016. Soufflé: On Synthesis of Program Analyzers. In *CAV (2) (LNCS)*, Vol. 9780. Springer, 422–430.
[23] Philipp Körner, Michael Leuschel, João Barbosa, Vítor Santos Costa, Verónica Dahl, Manuel V. Hermenegildo, José F. Morales, Jan Wielemaker, Daniel Diaz, and Salvador Abreu. 2022. Fifty Years of Prolog and Beyond. *Theory Pract. Log. Program.* 22, 6 (2022), 776–858.
[24] Alexander Shkapsky, Mohan Yang, and Carlo Zaniolo. 2015. Optimizing recursive queries with monotonic aggregates in DeALS. In *ICDE*. IEEE, 867–878.
[25] Yujiao Zhou, Bernardo Cuenca Grau, Yavor Nenov, Mark Kaminski, and Ian Horrocks. 2015. PAGOdA: Pay-As-You-Go Ontology Query Answering Using a Datalog Reasoner. *J. Artif. Intell. Res.* 54 (2015), 309–367.