The Temporal Vadalog System

Luigi Bellomarini
Bank of Italy, Rome, Italy

Livia Blasi
TU Wien, Vienna, Austria
Bank of Italy, Rome, Italy

Markus Nissl
TU Wien, Vienna, Austria

Emanuel Sallinger
TU Wien, Vienna, Austria
University of Oxford, Oxford, UK

Abstract

The recent resurgence of the Datalog language in the Knowledge Representation and Reasoning community has paved the way for a very promising proposal for temporal extension. DatalogMTL (Datalog with Metric Temporal Operators) is a language that offers a good trade-off between computational complexity and expressive power. However, existing implementations are still preliminary or prototypical. In this extended abstract, we give a brief overview of Temporal Vadalog, a system supporting reasoning over DatalogMTL programs built upon an engineered architecture and adopted in production scenarios in the financial setting.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Information systems → Database management system engines

Keywords and phrases temporal reasoning, Datalog, DatalogMTL

Digital Object Identifier 10.4230/LIPIcs.TIME.2025.15

Category Short Paper

Related Version This is an extended abstract of a paper previously published on Theory and Practice of Logic Programming.
Full Version: `https://doi.org/10.1017/S1471068425000018` [4]

Funding This work has been supported by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201, 10.47379/VRG18013, 10.47379/NXT22018]; and by the Austrian Science Fund (FWF) 10.55776/COE12.

1 Introduction

The adoption of Datalog-based systems for *Knowledge Representation and Reasoning* (KRR) and their growing application in production settings such as the financial space [1] are going hand in hand with a wave of research into extensions to Datalog, known as the Datalog$^{\pm}$ family [10]. The aim is to strike a good balance between computational complexity and high expressivity, that is, to incorporate the features needed for real-world applications. In this context, a key requirement in KRR is native support for time through the reasoning process.

The recent introduction of the extension of Datalog with *metric temporal operators*, named *DatalogMTL* [8], along with the good computational characteristics of its fragments [16, 17, 18], brought the potential of temporal reasoning to real-world applications, from transport and robotics, from healthcare to finance. However, the integration of such capabilities into a production-ready system requires functional and architectural characteristics that


© Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger;
licensed under Creative Commons License CC-BY 4.0

ensure rigorous and efficient implementation of the language in computation and memory footprints. To our knowledge, all currently existing DatalogMTL implementations remain at a prototypical stage or are designed primarily for exploratory research [7, 19].

Contribution. Motivated by the need to adopt temporal reasoning in high-stake financial settings for the Central Bank of Italy, in this extended abstract [4], we introduce the Temporal Vadalog System, the temporal extension to the Vadalog System, a state-of-the-art Datalog-based reasoner [2].

Overview. Due to space constraints, the remainder of the paper provides an example-based glimpse into the functional and non-functional approaches to implementing DatalogMTL in Vadalog, specifically in Section 2, focussing on joins, and in Section 3, which describes the architecture. Some preliminaries about DatalogMTL can be found in Appendix A. For a complete overview, the reader is referred to the full paper [4].

2 Time-Aware Joins and Time Series Operators by Example

To briefly illustrate the capabilities of the system, we proceed by example with a realistic – albeit simplified – case from the financial domain.

▶ **Example 1.** A Financial Intelligence Unit (FIU) is an agency responsible for collecting information about suspicious financial activity to support investigations into money laundering or terrorism financing. They aim to identify companies with abnormal spending behavior and investigate the corporate networks they belong to. We describe the scenario by a database $\mathcal{D}$ of temporally annotated facts and the following set Π of DatalogMTL rules.

$$\boxminus_{[0,3]} \mathit{suspiciousActivity}(C), \neg \diamondminus_{[0,7]} \mathit{markedAsSafe}(C) \rightarrow \mathit{directSuspicious}(C) \tag{1}$$

$$\mathit{directSuspicious}(C) \rightarrow \mathit{suspicious}(C) \tag{2}$$

$$\mathit{suspicious}(C), \mathit{linked}(C, O) \rightarrow \mathit{suspicious}(O) \tag{3}$$

We assume that the reader is familiar with the logic-based formulation of Datalog syntax, with the body of a rule (left-hand side) being the implication premise, a logical conjunction of predicate over terms (i.e., constants or variables), while the head (right-hand side) is the implication conclusion. Ignoring the temporal angle, Rule 1 describes the conditions (company C involved in *suspiciousActivity* while not being *markedAsSafe*) under which a company C is marked as *directSuspicious*, while Rules 2-3 recursively mark every *directSuspicious* company C as *suspicious* (Rule 2) and then proceed to mark every other company o that is *linked* to *suspicious* company C as *suspicious* as well (Rule 3). Coming back to Rule 1, we see how the temporal aspect is essential: a company that is *markedAsSafe* in a distant past is not thereby exempt from having its current *suspiciousActivity* scrutinized. Metric temporal operators are helpful here: assuming day granularity, when prefixed with $\boxminus_{[0,3]}$, the atom *suspiciousActivity* only holds if the atom itself has continuously held in the interval $[t-0, t-3]$ if evaluated at t – the *suspiciousActivity* continuously held for the previous 3 days – while with $\diamondminus_{[0,7]}$, *markedAsSafe* only holds if the clearance occurred sometime between $[t-0, t-7]$. Let us consider database $\mathcal{D}$:

$$\mathcal{D} =$$

$$\{\mathit{suspiciousActivity}(\mathrm{A})@[\mathtt{May}-\mathtt{01}, \mathtt{May}-\mathtt{04}], \mathit{markedAsSafe}(\mathrm{A})@[\mathtt{Apr}-\mathtt{30}, \mathtt{Apr}-\mathtt{30}],$$
$$\mathit{suspiciousActivity}(\mathrm{B})@[\mathtt{May}-\mathtt{02}, \mathtt{May}-\mathtt{05}], \mathit{markedAsSafe}(\mathrm{B})@[\mathtt{Apr}-\mathtt{21}, \mathtt{Apr}-\mathtt{21}]\}$$

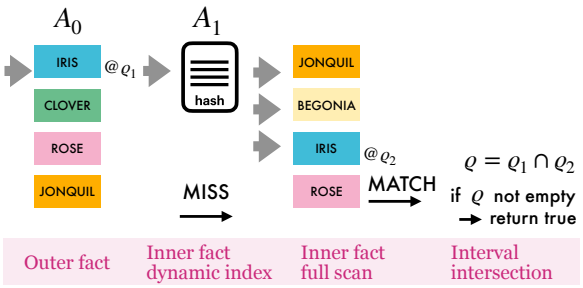

Figure 1 Illustration of an example of a temporal join algorithm execution; atoms with predicates A_0 and A_1 are joined on a *PlantNameTerm*.

Program:	**EMA**	
Dataset	Vadalog	InfluxDB
NQ1	0.0953	13.989
NQ10	4.8127	34.57
NQ50	74.1943	591.554
NQ100	306.6180	820.992

Figure 2 Experiment results, expressed in seconds, for the Exponential Moving Average in Vadalog and in the TSDB InfluxDB in the full paper [4]

By applying Rule 1, we see that only company B would be marked as *directSuspicious* as it was not *markedAsSafe* in the 7 days prior to the *suspiciousActivity*. However, to compute the conjunction between these two temporal atoms – $\boxminus_{[0,3]}$*suspiciousActivity* and $\diamondminus_{[0,7]}$*markedAsSafe* – one would need a *time-aware* join, able to join facts and their temporal intervals at the same time, while also handling stratified negation.

Temporal Join Algorithm

In Temporal Vadalog, the join algorithm is a temporal interval extension of the slot machine join of the Vadalog system [2], which is based on the index nested loop join [11]. In particular, during the first full scan of the inner relation, an in-memory index is constructed, unlike in the usual algorithms, where a precomputed index is typically present in materialized form.

The algorithm, fully reported in Algorithm 1, intuitively works as follows. Assume we have two predicates to be joined A_0 and A_1; first, we use the A_1 index to retrieve the next already-scanned fact that matches the join term(s) from A_0 (Line 8). If the index does not contain such a fact, we pursue the full scan until either a matching fact is found or all facts have been examined (Lines 10-14). If no further fact is found (Lines 15-22), we continue the scan with the next A_0 (if it exists and if A_1 is not negated). In case a fact is found, independently of whether it is from the index or the full scan, we produce the valid interval of the joined fact using the join logic (Line 23): difference for a negated literal, intersection for a positive interval, or a blend of interval operations and set operations for temporal operators like $\mathcal{S}$ (*since*) and $\mathcal{U}$ (*Until*). In the end, we check whether the resulting interval is valid, and if not, if A_1 is negated, we continue the scan with the next A_0 (if it exists) (Lines 25-29), otherwise we proceed with the loop; if the interval is non-empty and A_k is not negated, we return true as we have found a valid joined fact (Lines 31-32); otherwise, we continue retrieving the next "negated" fact. A visual representation of the execution of a temporal join is shown in Figure 1.

Time Series Operators

In Example 1, we assumed that the *suspiciousActivity* facts were provided by $\mathcal{D}$. Now we want to consider the predicate as a result of a different reasoning task.

▶ **Example 2.** Understanding whether a spending activity is suspicious implies detecting anomalous patterns, a task extremely relevant for financial authorities. A form of behavioural analysis is adopted here: if the number of flagged transactions is greater than the company average for a given period, then the spending activity is suspicious.

$$flagTransactions(C, AMT), sma(C, AVG), AMT > AVG \rightarrow suspiciousActivity(C) \quad (4)$$

Algorithm 1 Temporal Join between two predicates.

Input: predicates A_0 and A_1 to be joined, with A_0 not negated

```
1:  I_0 ← A_0.iterator()
2:  I_1 ← A_1.iterator()
3:  (a_0, ϱ_0) ← I_0.getNext()
4:  function NEXT
5:      interval ← ϱ_0
6:      while true do
7:          X ← a_0.joinTerm
8:          (a_1, ϱ_1) ← A_1.index.get(X)
9:          if a_1 is null then                          ▷ Continue full scan, if index miss
10:             while I_1.next() do
11:                 (a_1, ϱ_1) ← I_1.getNext()
12:                 A_1.index.put(a_1)                   ▷ Update the index map for A_1
13:                 if a_1.joinTerm == a_0.joinTerm then
14:                     break                            ▷ Exit the inner loop if matching fact is found
15:                 end if
16:             end while
17:         end if
18:         if a_1 is null then                          ▷ No further matching fact A_1 found
19:             if A_1 is negated then
20:                 return true
21:             end if
22:             if I_0.next() is false then
23:                 return false
24:             end if
25:             (a_0, ϱ_0) ← I_0.getNext()               ▷ Repeat loop for next a_0
26:             interval ← ϱ_0
27:             continue
28:         end if
29:         interval ← resultingIntervalFromJoinLogic(ϱ_1, interval)
30:         if interval is empty then
31:             if A_1 is negated then
32:                 if I_0.next() is false then
33:                     return false
34:                 end if
35:                 (a_0, ϱ_0) ← I_0.getNext()           ▷ Repeat loop for next a_0
36:                 interval ← ϱ_0
37:             end if
38:             continue
39:         end if
40:         if A_1 is not negated then
41:             return true
42:         end if
43:     end while
44: end function
```

The average value is given by the *sma* predicate (*simple moving average*), which encapsulates a time series operator that performs a moving average calculation to smooth the signal and filter out noise and transient variations. While time series analysis typically adopts ad-hoc software libraries, Temporal Vadalog intrinsically offers such statistics:

$$\diamondsuit_{[0,n)} timeSeries(X, Value) \rightarrow extended(X, Value) \tag{5}$$

$$timeSeries(X, Value), extended(X, Roll) \rightarrow rolling(X, Roll) \tag{6}$$

$$rolling(X, Roll), Avg = avg(Roll) \rightarrow sma(X, Avg) \tag{7}$$

We use $\diamondsuit$ to extend the validity of the window over n days (Rule 5), and we join it with the original time series to pin it to the correct starting date (Rule 6). As in SMA all data points have equal weight, Rule 7 computes the mathematical average over every window. Performance has been evaluated against a time series database (TSDB) in the full paper [4], in this case with the *exponential moving average* (EMA), on the NASDAQ Composite Index time series [14]. Results are shown in Figure 2.

3 The Temporal Vadalog Architecture

The temporal join is only one component of the system, and several others are required in order to support query answering over a set of rules like that of Examples 1-2, among which the transformations from the temporal operators and termination of recursive rules. Looking at the larger picture, the Temporal Vadalog architecture extends the *volcano iterator model* [13] of the Vadalog system [6] with *time-awareness*. A DatalogMTL program Π is transformed into an execution pipeline that reads data from sources, applies the transformations (both algebraic and time-related ones) and returns the intended output as a result. The process consists of two stages: in the first, the pipeline is built through a sequence of compilers and optimizers that gradually transforms the set of rules into a *reasoning query plan*. Taking inspiration from the *pipe and filters* architecture [9], each required transformation is represented by a filter, while dependencies between rules are represented by pipes. The second stage is at runtime, where a *pull-based* approach is used. Starting from the output filter, `next()` calls propagate through filter chains to source filters. Each filter applies the requested transformation based on the rule it represents. As long as data are available in the filter cascade, `next()` succeeds.

A number of time-relevant operations are tackled along the way: (a) the transformation of time intervals through the application of temporal operators; (b) the implementation of *merging operations* for intervals through various strategies to ensure correctness and efficiency [3]; (c) the temporal joins in the presence of *stratified negation*; (d) detection of repeating temporal patterns through the so-called *termination strategies*, i.e. techniques to guaranteee termination; (e) *temporal aggregations* [5]; and (f) the possibility to switch between temporal and to non-temporal reasoning, to activate non-temporal features, essential in some reasoning settings, such as *existential quantification*.

Summary. In this work, we showed the Temporal Vadalog system by first describing temporal joins and operators applied to an example, and concluded by briefly discussing the "big picture" of its architecture. The interested reader is referred to the full paper.

References

1 Teodoro Baldazzi, Luigi Bellomarini, and Emanuel Sallinger. Reasoning over financial scenarios with the Vadalog system. In *EDBT*, pages 782–791. OpenProceedings.org, 2023. doi:10.48786/edbt.2023.66.

2 Luigi Bellomarini, Davide Benedetto, Georg Gottlob, and Emanuel Sallinger. Vadalog: A modern architecture for automated reasoning with large knowledge graphs. *IS*, 105:101528, 2022. doi:10.1016/j.is.2020.101528.

3 Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger. The Temporal Vadalog system. In *RuleML+RR*, volume 13752, pages 130–145. Springer, 2022. doi:10.1007/978-3-031-21541-4_9.

4 Luigi Bellomarini, Livia Blasi, Markus Nissl, and Emanuel Sallinger. The Temporal Vadalog system: Temporal Datalog-based reasoning. *Theory and Practice of Logic Programming*, pages 1–29, 2025. doi:10.1017/S1471068425000018.

5 Luigi Bellomarini, Markus Nissl, and Emanuel Sallinger. Monotonic Aggregation for Temporal Datalog. In *Proceedings of the 15th International Rule Challenge*, volume 2956, 2021. URL: https://ceur-ws.org/Vol-2956/paper30.pdf.

6 Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vadalog System: Datalog-based reasoning for knowledge graphs. *PVLDB*, 11(9):975–987, 2018. doi:10.14778/3213880.3213888.

7 Sebastian Brandt, Elem Güzel Kalayci, Roman Kontchakov, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Ontology-based data access with a Horn fragment of metric temporal logic. In *AAAI*, pages 1070–76. AAAI Press, 2017. doi:10.1609/aaai.v31i1.10696.

8 Sebastian Brandt, Elem Güzel Kalayci, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Querying log data with metric temporal logic. *J. Artif. Intell. Res.*, 62:829–877, 2018. doi:10.1613/jair.1.11229.

9 Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-oriented software architecture, 4th Edition*. Wiley, 2007. URL: https://www.worldcat.org/oclc/314792015.

10 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Semant.*, 14:57–83, 2012. doi:10.1016/j.websem.2012.03.001.

11 Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.

12 Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991. doi:10.1145/116825.116838.

13 Goetz Graefe and William J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *ICDE*, pages 209–218, 1993. doi:10.1109/ICDE.1993.344061.

14 NASDAQ OMX Group. NASDAQ composite index. http://tinyurl.com/frednq, 2023. FRED, Federal Reserve Bank of St. Louis, Accessed: 2023-03-22.

15 David J. Tena Cucala, Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, and Egor V. Kostylev. Stratified negation in Datalog with metric temporal operators. In *AAAI*, pages 6488–6495, 2021. doi:10.1609/aaai.v35i7.16804.

16 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL: Computational complexity and expressive power. In *IJCAI*, pages 1886–1892, 2019. doi:10.24963/ijcai.2019/261.

17 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. DatalogMTL over the integer timeline. In *KR*, pages 768–77, 2020. doi:10.24963/kr.2020/79.

18 Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. Tractable fragments of Datalog with metric temporal operators. In *IJCAI*, pages 1919–1925, 2020. doi:10.24963/ijcai.2020/266.

19 Dingmin Wang, Pan Hu, Przemyslaw Andrzej Walega, and Bernardo Cuenca Grau. MeTeoR: Practical reasoning in Datalog with metric temporal operators. In *AAAI*, pages 5906–5913, 2022. doi:10.1609/aaai.v36i5.20535.

The appendix includes excerpts from the full version of the paper [4] to cover the more technical parts of this extended abstract. In particular, Appendix A provides the background of DatalogMTL.

A DatalogMTL

DatalogMTL is Datalog extended with operators from the metric temporal logic. We provide a summary of DatalogMTL with stratified negation under continuous semantics. DatalogMTL is defined over the rational timeline, i.e., an ordered set of rational numbers $\mathbb{Q}$. An interval $\varrho = \langle \varrho^-, \varrho^+ \rangle$ is a non-empty subset of $\mathbb{Q}$ such that for each $t \in \mathbb{Q}$ where $\varrho^- < t < \varrho^+$, $t \in \varrho$, and the endpoints $\varrho^-, \varrho^+ \in \mathbb{Q} \cup \{-\infty, \infty\}$. The brackets denote whether the interval is closed ("[]"), half-open ("[)","(]") or open ("()"), whereas angle brackets ("$\langle\rangle$") are used when unspecified. An interval is *punctual* if it is of the form $[t, t]$, *positive* if $\varrho^- \geq 0$, and *bounded* if $\varrho^-, \varrho^+ \in \mathbb{Q}$.

DatalogMTL extends the syntax of Datalog with negation with temporal operators [15]. For the following definitions, we consider a function-free first-order signature. An *atom* is of the form $P(\boldsymbol{\tau})$, where P is a n-ary predicate and $\boldsymbol{\tau}$ is a n-ary tuple of terms, where a term is either a constant or a variable. An atom is *ground* if it contains no variables. A *fact* is an expression $P(\boldsymbol{\tau})@\varrho$, where ϱ is an interval and $P(\boldsymbol{\tau})$ a ground atom and a *database* is a set of facts. A *literal* is an expression given by the following grammar, where ϱ is a positive interval: $A ::= \top \mid \bot \mid P(\boldsymbol{\tau}) \mid \boxminus_\varrho A \mid \boxplus_\varrho A \mid \diamondminus_\varrho A \mid \diamondplus_\varrho A \mid A \,\mathcal{S}_\varrho\, A \mid A \,\mathcal{U}_\varrho\, A$. A *rule* is an expression given by the following grammar, where $i, j \geq 0$, each A_k $(k \geq 0)$ is a literal and B is an atom: $A_1 \wedge \cdots \wedge A_i \wedge \text{not } A_{i+1} \wedge \cdots \wedge \text{not } A_{i+j} \rightarrow B$. The conjunction of literals A_k is the rule body, where $A_1 \wedge \cdots \wedge A_i$ denote positive literals and $A_{i+1} \wedge \cdots \wedge A_{i+j}$ denote negated (i.e., prefixed with *not*) literals. The atom B is the rule head. A rule is *safe* if each variable occurs in at least one positive body literal, *positive* if it has no negated body literals (i.e., $j = 0$), and *ground* if it contains no variables. A program Π is a set of safe rules and is *stratifiable* if there exists a stratification of a program Π. A stratification of Π is defined as a function σ that maps each predicate P in Π to a positive integer (*stratum*) s.t. for each rule, where P^h denotes a predicate of the head, and P^+ (resp. P^-) a positive (negative) body predicate, $\sigma(P^h) \geq \sigma(P^+)$ and $\sigma(P^h) > \sigma(P^-)$. The semantics of DatalogMTL is given by an interpretation $\mathfrak{M}$ that specifies for each time point $t \in \mathbb{Q}$ and each ground atom $P(\boldsymbol{\tau})$, whether $P(\boldsymbol{\tau})$ is satisfied at t, in which case we write $\mathfrak{M}, t \models P(\boldsymbol{\tau})$. This satisfiability notion extends to ground literals as follows:

$$
\begin{aligned}
&\mathfrak{M}, t \models \top && \text{for each } t \\
&\mathfrak{M}, t \models \bot && \text{for no } t \\
&\mathfrak{M}, t \models \boxminus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for all } s \text{ with } t - s \in \varrho \\
&\mathfrak{M}, t \models \boxplus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for all } s \text{ with } s - t \in \varrho \\
&\mathfrak{M}, t \models A \,\mathcal{S}_\varrho\, A' && \text{iff } \mathfrak{M}, s \models A' \text{ for some } s \text{ with } t - s \in \varrho \wedge\ \mathfrak{M}, r \models A \text{ for all } r \in (s, t) \\
&\mathfrak{M}, t \models A \,\mathcal{U}_\varrho\, A' && \text{iff } \mathfrak{M}, s \models A' \text{ for some } s \text{ with } s - t \in \varrho \wedge\ \mathfrak{M}, r \models A \text{ for all } r \in (t, s) \\
&\mathfrak{M}, t \models \diamondminus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for some } s \text{ with } t - s \in \varrho \\
&\mathfrak{M}, t \models \diamondplus_\varrho A && \text{iff } \mathfrak{M}, s \models A \text{ for some } s \text{ with } s - t \in \varrho
\end{aligned}
$$

An interpretation $\mathfrak{M}$ satisfies not A ($\mathfrak{M}, t \models \text{not } A$), if $\mathfrak{M}, t \not\models A$, a fact $P(\boldsymbol{\tau})@\varrho$, if $\mathfrak{M}, t \models P(\boldsymbol{\tau})$ for all $t \in \varrho$, and a set of facts $\mathcal{D}$ if it is a model of each fact in $\mathcal{D}$. Furthermore, $\mathfrak{M}$ satisfies a ground rule r if $\mathfrak{M}, t \models A_k$ for $0 \leq k \leq i$ and $\mathfrak{M}, t \models \text{not } A_k$ for $i+1 \leq k \leq i+j$ for every t; for every t, if the literals in the body are satisfied, so is the head $\mathfrak{M}, t \models B$; $\mathfrak{M}$

satisfies a rule when it satisfies every possible grounding of the rule. Moreover, $\mathfrak{M}$ is a *model* of a program if it satisfies every rule in the program and the program has a stratification, i.e., it is *stratifiable*. Given a stratifiable program Π and a set of facts $\mathcal{D}$, we call $\mathfrak{C}_{\Pi,\mathcal{D}}$ the *canonical model* of Π and $\mathcal{D}$ [8], and define it as the minimum model of Π and $\mathcal{D}$, i.e., $\mathfrak{C}_{\Pi,\mathcal{D}}$ is the minimum model for all the facts of $\mathcal{D}$ and the rules of Π. In this context, "minimum" means that the set of positive literals in $\mathfrak{M}$ is minimized or, equivalently, that the positive literals of this model are contained in every other model. Since Π is stratifiable, this minimum model exists and is unique [12]. According to Tena Cucala's notation [15], we say that a stratifiable program Π and a set of facts $\mathcal{D}$ entail a fact $P(\boldsymbol{\tau})@\varrho$, written as $(\Pi, \mathcal{D}) \models P(\boldsymbol{\tau})@\varrho$, if $\mathfrak{C}_{\Pi,\mathcal{D}} \models P(\boldsymbol{\tau})@\varrho$. In the remainder of the paper, we will assume the stratification of programs (or set of rules) as implicit.

In this context, the *query answering* or *reasoning* task is defined as follows: given the pair $Q = (\Pi, Ans)$, where Π is a set of rules, Ans is an n-ary predicate, and the query Q is evaluated over $\mathcal{D}$, then $Q(\mathcal{D})$ is defined as $Q(\mathcal{D}) = \{(\bar{t}, \varrho) \in dom(\mathcal{D})^n \times time(\mathcal{D}) \mid (\Pi, \mathcal{D}) \models Ans(\bar{t})@\varrho\}$, where $\bar{t}$ is a tuple of terms, the domain of $\mathcal{D}$, denoted $dom(\mathcal{D})$, is the set of all constants that appear in the facts of $\mathcal{D}$, and the set of all the time intervals in $\mathcal{D}$ is denoted as $time(\mathcal{D})$. As we shall see in practical cases, the Ans predicate of Π will be sometimes called "query predicate" and provided to the reasoning system with specific conventions, which we omit for space reasons, but will render in textual explanations.