Theoretical characterization of Signal Diagnostic Processing Language

Ognjen Savković[1], Evgeny Kharlamov[2], Guohui Xiao[1], Gulnar Mehdi[3,5], Elem Güzel Kalaycı[1], Werner Nutt[1], Mikhail Roshchin[3], and Ian Horrocks[2]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
[2] University of Oxford, Oxford, UK
[3] Siemens AG, Corporate Technology, Munich, Germany

Abstract. In this paper we analyze the theoretical aspects of our semantic rule language, SDRL. Inspired with an industrial collaboration, SDRL is designed to process signals from sensors installed in industrial equipment by filtering, aggregating, and combining sequences of time-stamped measurements recorded by the sensors. In our previous work, we conducted extensive experiments showing scalability of the language on real data. Here instead, we analyze theoretical properties of SDRL. In particular, we show that query answering in SDRL can be reduced to relational algebra with aggregations. To this end, we rely on recently introduced $Datalog_{nr}MTL$ that is first order rewritable. First we introduce an extension of $Datalog_{nr}MTL$ with aggregate functions. Then we show how to translate query answering in SDRL into the query answering in the extended language.

1 Introduction

Large engineering and maintenance companies like Siemens typically rely on *rule-based* intelligent diagnostics: diagnostic engineers create and use complex diagnostic rule-sets to detect abnormalities during equipment run time and sophisticated analytical models to combine these abnormalities with models of physical aspects of equipment such as thermodynamics and energy efficacy [1].

An important class of rules that are commonly used in Siemens for rule-based equipment diagnostics are *signal processing rules* (SPRs). SPRs allow to filter, aggregate, combine, and compare *signals*, that are time stamped measurement values, coming from sensors installed in equipment and trigger error or notification messages when a certain criterion has been met. Thus, sensors report temperature, pressure, vibration and other relevant parameters of equipment and SPRs process this data and alert whenever a certain pattern is detected. Moreover, SPRs can combine data of two types: *master data* that contains train engineering specifications, results of previous diagnostic tasks, and diagnostic event data, and *operational data* that contains recorded signals from sensors installed in equipment. Data processed by SPRs is then used by analytical models in order to ensure safety, efficiency, and to lower the maintenance costs.

SPRs that are currently offered in most of existing diagnostic systems and used in Siemens are highly *data dependent* in the sense that *(i)* data about specific characteristics like speed, capacity, and identifiers of individual sensors and pieces of equipment are explicitly encoded in SPRs and *(ii)* schema of such data is reflected in the SPRs. As a

result, for a typical simple train diagnostic task an engineer has to write from dozens to hundreds of SPRs with hundreds of sensor tags, component codes, sensor and threshold values and equipment configurations and design data.

Example 1. Consider the diagnostic task: *all door sensors indicate that the car doors are OK, and the main train line pressure is trending upwards for more than 33 seconds.*

The above task can be checked by analyzing the train sensors. For instance, let us assume that the locomotive L1 has door sensors SKNF_X01, SKNF_X02, SKNF_X03 and SKNF_X04, and let MNPRSS be the sensor that measures the main train pressure. Further, let us assume that the task is verified as *OK* if the sum of the above sensors values is equal to 16. Then, the following data dependent SPR written in a syntax similar to the one of Siemens SPRs can be used to accomplish the task:

$$\begin{aligned} \$\mathsf{CarDoorsOK} = \texttt{truth}\,(\texttt{sum}\,(\mathsf{SKNF\_X01}, \mathsf{SKNF\_X02}, \mathsf{SKNF\_X03}, \mathsf{SKNF\_X04}) \quad (1)\\ : \texttt{value}(=, 16)\ \&\&\texttt{trend}(\mathsf{MNPRSS}, \textit{'up'}) : \texttt{duration}(>, 0.33s)) \end{aligned}$$

Here $CarDoorsOK is a Boolean variable. Many of these SPRs differ only on specific sensor identifiers and the number of speed signals to aggregate. Adapting these SPRs to another cars will also require to change a lot of identifiers.

Data dependence of SPRs poses significant challenges for diagnostic engineers in *(i)* authoring, and then *(ii)* reuse, and *(iii)* maintenance of SPRs. These challenges are common for large enterprises and Siemens is not an exception.

In our previous work [2, 3], we proposed a higher-level rule language *Semantic Diagnostic Rule Language* (*SDRL*) that allows to write SPRs and complex diagnostic tasks in an abstract fashion by exploiting both ontological vocabulary and queries over ontologies to identify relevant sensors and data values. We implemented *SDRL* in a prototypical system and deployed an implementation in Siemens [4]. Also, we evaluated usability of our solution with Siemens engineers by checking how fast they are in formulating diagnostic tasks in *SDRL*. We also evaluated the efficiency of our solution in processing diagnostic tasks over train and turbine signals in a controlled environment.

Contributions. In this work, we study the formal properties of the *SDRL* language.

We prove that the reasoning tasks in *SDRL* programs can be rewritten into query answering using linear algebra with aggregates (that is SQL). Further, we analyzed the upper bound of the complexity of our problem. In particular, we measure the complexity of the problem in the size of two main components: the program and the data. We expect, the size of the data largely dominates the size of programs, and while the data can be huge (typically several GBs) that the size of program can be significantly big (e.g., several thousands of rules) thus both measures are relevant. [4]

To study the two properties from above, we encode the problem of firing a message in *SDRL* into the fact-entailment problem [6] over an extended version of the recently introduced non-recursive metric Datalog, Datalog$_{\text{nr}}$*MTL* [7]. The reason for doing this is twofold. First, Datalog$_{\text{nr}}$*MTL* (inspired by the well-studied *Metric Temporal Logic*

[4] We point out that the notion of "rewriting into linear algebra with aggregates" is very similar to the formal property of First-order rewritability (e.g., see [5]) but since the latter does not consider aggregates we use the former.

(MTL) [7]) provides a natural way to model rules that reason over time intervals. Second, Datalog$_{\text{nr}}$*MTL* is a language suitable for OBDA [5, 8], that is, it has been shown how to rewrite queries over the rules in Datalog$_{\text{nr}}$*MTL* into standard SQL over the sources [7].

Still, Datalog$_{\text{nr}}$*MTL* cannot be immediately related to our language since it does not support aggregates and some other logic constructs that we need for our encoding (in particular, functional symbols, negation and aggregates). So first, we extend DatalogMTL with functional symbols, aggregation, etc. under reasonable restrictions, without increasing the complexity. Then to show that our problem is rewritable into linear algebra with aggregates we do the following encoding. Given a program Π and a message rule r we create an extended non-recursive Datalog$_{\text{nr}}$*MTL* program $\Sigma_{\Pi,r}$ and a proposition m_r only such that: Π fires r iff Σ_Π "entails" m_r. A corollary of this gives us *(i)* that our language is suitable for OBDA [5] specification (follows from the encoding); *(ii)* ways to reformulate our programs and rules into SQL queries (extending the principles in [7]).

2 Signal Diagnostic Processing Language *SDRL*

In this section, we introduce formally the syntax and semantics of our signal diagnostic processing language, *SDRL*. To do so, we first introduce signal processing expressions that allow one to manipulate basic signals using mathematical functions. Then we introduce a notion of programs that allow one to compose and combine expressions, and form message rules. Finally, we provide semantics of our language that formally defines how *SDRL* should be executed.

2.1 Signals Processing Expressions

We introduce signal expressions that filter and manipulate basic signals and create new more complex signals. Intuitively, in our language we group signals in ontological concepts and signal expression are defined on the level of concepts. Then, a *signal processing expression* is recursively defined as follows:

$$
\begin{array}{rcll}
C & = & Q & \mid \{s_1, \ldots, s_m\} \mid \\
 & & \alpha \circ C & \mid C_1 : \textit{value}(\odot, \alpha) \mid \\
 & & \mathsf{agg}\ C_1 & \mid C_1 : \textit{duration}(\odot, t) \mid \\
 & & C_1 :\ \textit{align}\ C_2 & \mid C_1 : \textit{trend}(\textit{direction}).
\end{array}
$$

where C is a concept, Q is a CQ with one output variable, $\circ \in \{+, -, \times, /\}$, $\mathsf{agg} \in \{\mathsf{min}, \mathsf{max}, \mathsf{avg}, \mathsf{sum}\}$, $\alpha \in \mathbb{R}$, $\odot \in \{<, >, \leq, \geq\}$, $\mathit{align} \in \{\textit{within}, \textit{after}[t], \textit{before}[t]\}$, t is a period, and $\textit{direction} \in \{\mathsf{up},\ \mathsf{down}\}$. In the following we will consider *well formed* sets of signal expressions, that is, sets where *(i)* each concept is defined at most once and *(ii)* where definitions of new concepts are assumed to be acyclic: if C_1 is used to define C_2 (directly or indirectly) then C_1 cannot be defined (directly or indirectly) using C_2.

Expressions $C = Q$ and $C = \{s_1, \ldots, s_m\}$ we call *basic signal expressions* and other we call *complex signal expressions*.

The formal meaning of signal processing expressions is defined in Table 1.

Example 2. The data-driven rules that can be used to determine that car doors function well, as in Equation (1), can be expressed with two concepts in *SDRL* as follows:

$$\mathsf{DoorsLocked} = \textit{sum}\ \mathsf{MainCarDoors} : \textit{value}(=, \textit{LockedValue}) \quad (2)$$

$$\mathsf{PressureUp} = \mathsf{CabinPressure} : \textit{trend}(\textit{'up'}) : \textit{duration}(>, 33sec) \quad (3)$$

Here, MainCarDoors is a CQ defined over a database. For brevity we do not introduce a new concept for each expression but we just join them with symbol ":". The constant *LockedValue* is a parameter for analyzing door of a train, and they are instantiated from the train configuration when the expressions are evaluated.

2.2 Diagnostic Programs and Messages

We now show how to use signal expressions to compose diagnostic programs and to alert messages.

A *diagnostic program* (or simply *program*) Π is a tuple $(\mathcal{S}, \mathcal{K}, \mathcal{M})$ where $\mathcal{S}$ is a set of basic signals, $\mathcal{K}$ a KB, $\mathcal{M}$ a set of well formed signal processing expressions such that each concept that is defined in $\mathcal{M}$ does not appear in $\mathcal{K}$.

In order to enjoy favorable semantic and computational characteristics of OBDA, we consider well-studied ontology language OWL 2 QL that formal basis is *DL-Lite*$_R$ [5].

On top of diagnostic programs Π *SDRL* allows to define *message rules* that report the current status of a system. Formally, they are defined as Boolean combinations of signal processing expressions:

$$D := \quad C \quad | \quad \textit{not}\ D_1 \quad | \quad D_1\ \textit{and}\ D_2.$$

A *message rule* is a rule of the form, where D is a concept and m is a (text) message:

$$\textit{message}(m) = D.$$

Example 3. Using Equations (2)–(3) we define the following message:

$$message(\textit{"All car doors OK"}) = \mathsf{DoorsLocked}\ \textit{and}\ \mathsf{PressureUp}.$$

The message intuitively indicates that the doors are functioning and locked.

Now we are ready to define the semantics of the rules, expression and programs.

2.3 Semantics of *SDRL*

We now define how to determine whether a program Π fires a rule r. To this end, we extend first-order interpretations that are used to define semantics of OWL 2 KBs.

Formally, our *interpretation* $\mathcal{I}$ is a pair $(\mathcal{I}_{FOL}, \mathcal{I}_{\mathcal{S}})$ where $\mathcal{I}_{FOL}$ interprets objects and their relationships (like in OWL 2) and $\mathcal{I}_{\mathcal{S}}$—signals. First, we define how both components of $\mathcal{I}$ interprets basic signals in $\mathcal{S}$. Formally, $\mathcal{S}^{\mathcal{I}} = \{s_1^{\mathcal{I}}, \ldots, s_n^{\mathcal{I}}\}$ where $\mathcal{I}_{FOL}$ 'returns' the signal id, i.e. $s^{\mathcal{I}_{FOL}} = o_s$ and $\mathcal{I}_{\mathcal{S}}$ 'returns' the signal itself, i.e. $s^{\mathcal{I}_{\mathcal{S}}} = s$.

Now we can define how $\mathcal{I}$ interprets KBs. Interpretation of a KB $\mathcal{K}^{\mathcal{I}}$ extends the notion of first-order logics interpretation as follows: $\mathcal{K}^{\mathcal{I}_{FOL}}$ is a first-order logics interpretation $\mathcal{K}$ and $\mathcal{K}^{\mathcal{I}_{\mathcal{S}}}$ is defined for objects, concepts, roles and attributes extending $S^{\mathcal{I}}$. That is, for each object o we define $o^{\mathcal{I}_{\mathcal{S}}}$ as s if o is the id of s from $\mathcal{S}$; otherwise

$C =$	Concept C contains
Q	all signal ids return by Q evaluated over the KB.
$\alpha \circ C_1$	new signal s' for each signal s in C_1 with $f_{s'} = \alpha \circ f_s$.
$C_1 : \mathit{value}(\odot, \alpha)$	new signal s' for each signal s in C_1 with $f_{s'}(t) = \alpha \odot f_s(t)$ if $\alpha \odot f_s(t)$ at time point t; otherwise $f_{s'}(t) = \bot$.
$C_1 : \mathit{duration}(\odot, t')$	new signal s' for each $s \in C_1$ with $f_{s'}(t) = f_s(t)$ if exists an interval I s.t.: f_s is defined I, $t \in I$ and $\mathit{size}(I) \odot t'$; otherwise $f_{s'}(t) = \bot$.
$\{s_1, \ldots, s_m\}$	all enumerated signal $\{s_1, \ldots, s_m\}$.
$\mathsf{agg}\ C_1$	new signal s' with $f_{s'}(t) = \mathsf{agg}_{s \in C_1} f_s(t)$, that is, s' is obtained from all signals in C_1 by applying the aggregate agg at each time point t.
$C_1 : \mathit{align}\ C_2$	new signal s_1 from C_1 if: exists a signal s_2 from C_2 that is *aligned* with s_1, i.e., for each interval I_1 where f_{s_1} is defined there is an interval I_2 where f_{s_2} is defined s.t. I_1 *aligns* with I_2.
$C_1 : \mathit{trend}(\mathit{direction})$	one signal s' for each signal s in C_1 with $f_{s'}(t) = f_s(t)$ if exists an interval I around t s.t.: f_s is defined I, and f_s is an increasing (decreasing) function on I for *direction=up* (=*down* resp.)

Table 1. Meaning of signal processing expressions. For the interval I, size(I) is its size. For intervals I_1, I_2 the *alignment* is: "I_1 *within* I_2" if $I_1 \subseteq I_2$; "I_1 *after*$[t]$ I_2" if all points of I_2 are after I_1 and the start of I_2 is within the end of I_1 plus period t; "I_1 *before*$[t]$ I_2" if "I_2 *start*$[t]$ I_1". In order to make the mathematics right, we assume that $c \circ \bot = \bot \circ c = \bot$ and $c \odot \bot = \bot \odot c = \mathit{false}$ for $c \in \mathbb{R}$, and analogously we assume for aggregate functions. If the value of a signal function at a time point is not defined with these rules, then we define it as $\bot$.

$(o, f_\bot)$. Then, for a concept A we define $A^{\mathcal{I}_\mathcal{S}} = \{s^{\mathcal{I}_\mathcal{S}} \mid o_s^{\mathcal{I}_{FOL}} \in A^{\mathcal{I}_{FOL}}\}$. Similarly, we define $\cdot^{\mathcal{I}_\mathcal{S}}$ for roles and attributes.

Finally, we are ready to define $\mathcal{I}$ for signal expressions and we do it recursively following the definitions in Table 1.

We now illustrate some of them. For example, if $C = \{s_1, \ldots, s_m\}$, then $C^{\mathcal{I}} = \{s_1^{\mathcal{I}}, \ldots, s_m^{\mathcal{I}}\}$; if $C = Q$ then $C^{\mathcal{I}_{FOL}} = Q^{\mathcal{I}_{FOL}}$ where $Q^{\mathcal{I}_{FOL}}$ is the evaluation of Q over $\mathcal{I}_{FOL}$ and $C^{\mathcal{I}_\mathcal{S}} = \{s \mid o_s^{\mathcal{I}_{FOL}} \in Q^{\mathcal{I}_{FOL}}\}$, provided that $\mathcal{I}_{FOL}$ is a model of $\mathcal{K}$. Otherwise we define $C^{\mathcal{I}} = \emptyset$. Similarly, we define interpretation of the other expressions.

Firing a Message Let Π be a program and 'r : *message*$(m) = C$' a message rule. We say that Π *fires* message r if *for each* interpretation $\mathcal{I} = (\mathcal{I}_{FOL}, \mathcal{I}_\mathcal{S})$ of Π it holds $C^{\mathcal{I}_{FOL}} \neq \emptyset$, that is, the concept that fires r is not empty. Our programs and rules enjoy the *canonical* model property, that is, each program has a unique (Hilbert) interpretation [9] which is minimal and can be constructed starting from basic signals and ontology by following signal expressions. Thus, one can verify $C^{\mathcal{I}_{FOL}} \neq \emptyset$ only on the canonical model and thus one can evaluate *SDRL* programs and expressions in a bottom-up fashion.

3 Extended Datalog*MTL*

In this part we introduce our extension of Datalog*MTL*. An atom A in extended Datalog*MTL* is either a comparison (e.g., $\tau \leq \tau'$) or defined by the following grammar:

$$A ::= P(\tau_1, \ldots, \tau_m) \mid \top \mid \boxplus_\varrho A \mid \boxminus_\varrho A \mid \diamondplus_\varrho A \mid \diamondminus_\varrho A$$
$$A\, \mathcal{U}_\varrho\, A' \mid A\, \mathcal{S}_\varrho\, A' \ \neg A \mid \tau = \mathsf{agg}[\![\tau_i \mid P(\tau_1, \ldots, \tau_m)]\!]$$

Here, P is a predicate, ϱ is an interval in reals, τ is a term (possibly with functional symbols), $\mathsf{agg} \in \{\mathsf{min}, \mathsf{max}, \mathsf{avg}, \mathsf{sum}\}$ and brackets $[\![\cdot]\!]$ denote multiset (values can

repeat). A *DatalogMTL program* Σ is a finite set of *rules* of the following form:

$$A^+ \leftarrow A_1 \wedge \cdots \wedge A_k \qquad \text{or} \qquad \bot \leftarrow A_1 \wedge \cdots \wedge A_k,$$

where A^+ is an atom that *does not* contain any 'non-deterministic' operators $\boxminus_\varrho, \diamondminus_\varrho, \mathcal{U}_\varrho$, $\mathcal{S}_\varrho$, negated atoms, or aggregate operators.

For our purposes it is sufficient to consider non-recursive programs. Informally, that are programs where dependency (direct or indirect) between predicates is acyclic. In fact, it is not trivial to understand how one would even define the semantics of programs with recursion in case of aggregates and negation. So, from now, we only consider extended Datalog*MTL* programs that are non-recursive.

In Datalog*MTL*, temporal operators are defined over intervals and they take the form $\boxplus_\varrho, \boxminus_\varrho$ and $\mathcal{U}_\varrho$, which refer to the future, and $\boxminus_\varrho, \diamondminus_\varrho$ and $\mathcal{S}_\varrho$, which refer to the past where ϱ is an interval. For example, $\boxplus_\varrho A$ is true at t iff an atom A is true in all points of an interval ϱ in the future from t, while $\diamondminus_\varrho A$ is true at t iff there exists a point in the past not longer than ϱ from t where A is true. For the complete semantics of the temporal operators and rules, please see [7].

A (temporal) *data instance* is a finite set of *facts* of the form $P(c)@\iota$, where $P(c)$ is a ground atom and ι an interval. The fact $P(c)@\iota$ states that $P(c)$ holds throughout the interval ι. Moreover, we simply write $P(c)@t$ for $P(c)@[t,t]$.

Finally, every satisfiable extended *Datalog$_{nr}$MTL* program Σ with database $\mathcal{D}$ has the *canonical* (or *minimal*) *model of Π and $\mathcal{D}$*, $\mathfrak{M}_{\Pi,\mathcal{D}}$. As usual, the most important property of canonical model is that if a fact holds in canonical model then it holds in any other model.

4 Encoding into Extended DatalogMTL

We start with an example of the encoding then we show the complete encoding.

Example 4 (Example of Encoding). The query in signal processing expression in Examples 2 and 3 can be encoded in a modular way, starting from simpler to more complex expressions. We start with the encoding rules for Example 2. First, we show how to capture the expression "*sum* MainCarDoors" in (2). For that we use the following rule:

$$\mathsf{SumMainCarDoors}(c_{ar}), \mathsf{value}(c_{ar}, v_1) \leftarrow sum[\![v \mid \mathsf{MainCarDoors}(x), \mathsf{value}(x, v)]\!] = v_1.$$

Intuitively, the rule introduces a new constant c_{ar} representing the "aggregated main car door sensor" and assign the average value of all *main car door* sensors to it. Then, we encode the second part of (2), $value(=, \mathit{LockedValue})$, using SumMainCarDoors with:

$$\mathsf{DoorsLocked}(x) \leftarrow \mathsf{SumMainCarDoors}(x), \mathsf{value}(x, v), v = \mathsf{LockedValue}.$$

To encode expression (3) in Example 2 we need to use temporal operators. In particular, to encode CabinPressure : *trend('up')* we need to copy all intervals of a signal in CabinPressure on which the signal is trending up. For that we need universal quantification ("$\forall$"). This is expressible in Datalog by two rules connected with a negation. First we compute intervals on which a signal is not trending up with the rule:

$$\text{notTrendUp}_{\mathsf{CP}}(x) \leftarrow \mathsf{CabinPressure}(x), \mathsf{value}(x, v_1), \diamondminus_{(0,\delta]}(\mathsf{value}(x, v_2), v_1 > v_2)$$

Intuitively, formula $(\text{value}(x, v_1), \diamondminus_{(0,\delta]}(\text{value}(x, v_2), v_1 > v_2))$ evaluates to true for some value v_1 at a time point t if there exists an interval of a size at most δ containing t in which signal x has another value v_2 that is smaller than v_1. Here, a parameter δ is a "small" real number and it is typically selected based on the size of signal sampling.

We compute the trending-up intervals by eliminating non-trending-up time points:

$$\mathsf{CabinPressureAux}(f_{\mathsf{cp}}(x)) \leftarrow \mathsf{CabinPressure}(x), \neg\text{notTrendUp}_{\mathsf{CP}}(x)$$

Here, the functional symbol f_{cp} is used to create a new signal identifier for each x. The values of the new signals are the same as originals and they are just copied for each time-point that is "trending up":

$$\text{value}(f_{\mathsf{cp}}(x), v) \leftarrow \mathsf{CabinPressureAux}(f_{\mathsf{cp}}(x)), \text{value}(x, v)$$

To encode the construct *duration* we also need temporal operators. In particular, we encode the construct : $\textit{duration}(>, 33sec)$ with the rule:

$$\mathsf{PressureUp}(f_{\mathsf{pu}}(x)) \leftarrow \diamondminus_{[0,33s]} \boxplus_{[0,33s]} \mathsf{CabinPressureAux}(x)$$

Intuitively, the temporal operator $\boxminus_{[0,33s]}$ selects "an event that lasts for the last 33s", and the temporal operator $\diamondminus_{[0,33s]}$ selects "an event happens within the last 33s". The nesting $\diamondminus_{[0,33s]}\boxminus_{[0,33s]}$ of these two operators selects the whole duration of all the events lasting at least 33s. Similarly as above, the value is transferred with the rule:

$$\text{value}(f_{\mathsf{pu}}(x), v) \leftarrow \mathsf{PressureUp}(f_{\mathsf{pu}}(x)), \text{value}(x, v)$$

Finally, to encode message firingfrom Example 3 we introduce two propositionals p_{dl} and p_{pu} for concepts DoorsLocked and PressureUp, respectively. In particular, p_{dl} is true if there exists a signal in DoorsLocked that has at least one value. And similarly for p_{pu}. This is encoded with the rules:

$$p_{\mathsf{dl}} \leftarrow \diamondminus_{[0,\infty)}\mathsf{DoorsLocked}, \qquad p_{\mathsf{dl}} \leftarrow \diamondplus_{[0,\infty)}\mathsf{DoorsLocked},$$
$$p_{\mathsf{up}} \leftarrow \diamondminus_{[0,\infty)}\mathsf{PressureUp}, \qquad p_{\mathsf{up}} \leftarrow \diamondplus_{[0,\infty)}\mathsf{PressureUp}$$

Here, $\diamondminus_{[0,\infty)}$ and $\diamondplus_{[0,\infty)}$ are used to check if DoorsLocked has at least one signal with value in the past or in the future, respectively. Then we encode with the firing a message:

$$\textit{message}(\textit{"All car doors OK"}) \leftarrow p_{\mathsf{dl}}, p_{\mathsf{up}}$$

4.1 Full Specification of Encoding into Extended *Datalog$_{nr}$MTL*

Let $(\mathcal{S}, \mathcal{K}, \mathcal{M})$ be an *SDRL* program. We define a corresponding extended *Datalog$_{nr}$MTL* program $(\mathcal{D}_{\mathcal{S},\mathcal{K}}, \Pi_{\mathcal{M}})$ where the temporal facts $\mathcal{D}_{\mathcal{S},\mathcal{K}}$ encodes $\mathcal{S}$ and $\mathcal{K}$, and the program $\Pi_{\mathcal{M}}$ encodes expressions $\mathcal{M}$. In particular, for each basic signal $s = (o_s, f_s)$ in $\mathcal{S}$:

- if $f_s(t) = v$ we add $\textit{value}(s, v)@t$ to $\mathcal{D}_{\mathcal{S},\mathcal{K}}$, and
- if o_s is an answer of Q over KB $\mathcal{K}$ then we add $Q(o_s)@(-\infty, +\infty)$ to $\mathcal{D}_{\mathcal{S},\mathcal{K}}$.

We observe that the signals can be encoded as a finite database instance is due to the assumption that signals are step functions.

The program $\Pi_{\mathcal{M}}$ is constructed from $\mathcal{M}$ following the encodings in Table 2. The encoding is obtained by using a unary predicate τ_C for each signal processing expression C and a binary predicate *value* which we describe in the next paragraphs. It is important to note that these predicates are interpreted not like FO-predicates but using point based semantics (e.g., $\tau_C(o)$ is true or false for a constant *o at a given time point* t). For detailed semantics of such rules see [7].

More formally, for a signal $s = (o_s, f_s)$, the fact $\tau_C(o_s)$ is true at a time point t iff *(i)* $o \in C^{\mathcal{I}}$ and *(ii)* $f_s(t)$ is a real number. Condition *(ii)* simplifies the encoding since we do not need to define when a signal does not have a real value at a point; otherwise we have to have the rules that encode the absence of a real value. Further, we use functional symbols, e.g., f_C, to generate fresh signal identifier. E.g., for a signal s, $f_C(o_s)$ represents a new signal id obtained from s for the expression C.

To store the value of a signal at a time point we use the predicate *value*. That is, value(o_s, v) is true at point t iff $f_s(t) = v$.

The encoding rules for *trend*(up) and *trend*(down) are based on intervals. For them we introduce a parameter δ, a "small" real number, that we use to select an interval around a time point. In theory, such parameter should converge to 0 to indeed check the trend of a real function (in fact, one needs the first derivative), however, in practice we expect that one can select such δ a priori (e.g., the length of signal sampling since signals are step functions) that is sufficiently small to check the trend of a function for a particular time point.

It is easy to observe that the extended Datalog*MTL* program Σ_{Π} is non-recursive.

5 Formal Properties

In this part we state the formal properties of the encoding. First we introduce two lemmas that characterize the encodings of auxiliary predicates *value*, τ_C's and propositional p_D's. Then we use them to show the main encoding theorem.

The following lemma establishes the correspondence between a program and auxiliary predicates *value* and τ_C's.

Lemma 1. *Let $\Pi = (\mathcal{S}, \mathcal{K}, \mathcal{M})$ be an SDRL program and $\Sigma_{\Pi} = (\mathcal{D}_{\mathcal{S},\mathcal{K}}, \Pi_{\mathcal{M}})$ be the extended DatalogMTL program as defined above. Further, let $\mathcal{I}$ be the canonical interpretation for Π and let $\mathfrak{M}$ be the canonical interpretation for Σ_{Π}. Then, for a signal processing expression C and a time point t the following are equivalent:*

- $s^{\mathcal{I}} \in C^{\mathcal{I}}$ *and* $f_s(t) = v$*;*
- $\mathfrak{M}, t \models \tau_C(o_s)$ *and* $\mathfrak{M}, t \models$ *value*(o_s, v)*.*

Proof. The proof (in the both directions of "iff") is based on induction on the number of rules that are required to generate expression C starting from basis expressions. We show direction "$\Leftarrow$". The opposite one can be shown analogously.

Induction Base: In this case, C is defined either with $C = Q$ or $C = \{s_1, \ldots, s_m\}$.

Let us assume $C = Q$, and $s^{\mathcal{I}} \in C^{\mathcal{I}}$ and $f_s(t) = v$. Since s is a basic signal, $\mathcal{D}_{\mathcal{S},\mathcal{K}}$ must contain the fact value$(o_s, v)@t$. Moreover $\mathcal{K} \models Q(o_s)$ hence according to the rule

Expression C	Encoding of C
Q	$\tau_C(x) \leftarrow Q(x), \text{value}(x, v).$
$\{s_1, \ldots, s_m\}$	$\tau_C(s_i) \leftarrow \text{value}(s_i, v)$, for each s_i
$\alpha \circ C_1,$	$\tau_C(f_C(x)), \text{value}(f_C(x), v) \leftarrow \tau_{C_1}(x), \text{value}(x, v'), v = \alpha \circ v'.$
$C_1 : \mathit{value}(\odot, \alpha)$	$\tau_C(f_C(x)), \text{value}(f_C(x), v) \leftarrow \tau_{C_1}(x), \text{value}(x, v), v \odot \alpha.$
$C_1 : \mathit{duration}(\geq, t)$	$\tau_C(f_C(x)) \leftarrow \diamondsuit_{[0,t]} \boxplus_{[0,t]} \tau_{C_1}(x).$ $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$
$C_1 : \mathit{duration}(<, t)$	$\tau_C(f_C(x)) \leftarrow \tau_{C_1}(x),\ \neg\, (\diamondsuit_{[0,t]}(\boxplus_{[0,t]} \tau_{C_1}(x)).$ $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$
$\mathsf{agg}\ C_1$	$\tau_C(c), \text{value}(c, v) \leftarrow v = \mathsf{agg}[\![v_1 \mid \text{value}(x, v_1), \tau_{C_1}(x)]\!],$ where c is a fresh constant, $\mathsf{agg}[\![\cdot]\!]$ is an aggregation operator over bags
$C_1 : \mathit{after}[t]\ C_2$	$\tau_C(f_C(x_1)) \leftarrow \big(\tau_{C_1}(x_1)\big)\, \mathcal{U}_{[0,\infty)}\, \big((\neg\tau_{C_1}(x_1) \wedge \neg\tau_{C_2}(x_2))\, \mathcal{U}_{[0,t]}\, \tau_{C_2}(x_2)\big).$ $\text{value}(f_C(x_1), v) \leftarrow \tau_C(f_C(x_1)), \text{value}(x_1, v)$
$C_1 : \mathit{before}[t]\ C_2$	$\tau_C(f_C(x_1)) \leftarrow \big(\tau_{C_1}(x_1)\big)\, \mathcal{S}_{[0,\infty)}\, \big((\neg\tau_{C_1}(x_1) \wedge \neg\tau_{C_2}(x_2))\, \mathcal{S}_{[0,t]}\, \tau_{C_2}(x_2)\big).$ $\text{value}(f_C(x_1), v) \leftarrow \tau_C(f_C(x_1)), \text{value}(x_1, v)$
$C_1 : \mathit{within}\ C_2$	$\tau_C(f_C(x_1)) \leftarrow \Big(\big(\tau_{C_1}(x_1) \wedge \tau_{C_2}(x_2)\big)\, \mathcal{S}_{[0,\infty)}\, (\neg\tau_{C_1}(x_1))\Big)\ \mathcal{U}_{[0,\infty)}\ \big(\neg\tau_{C_1}(x_1)\big).$ $\text{value}(f_C(x_1), v) \leftarrow \tau_C(f_C(x_1)), \text{value}(x_1, v).$
$C_1 : \mathit{trend}(\mathsf{up})$	$\tau_C(f_C(x)) \leftarrow \tau_{C_1}(x), \neg\, \text{notTrendUp}_{C_1}(x)$ $\text{notTrendUp}_{C_1}(x) \leftarrow \tau_{C_1}(x), \text{value}(x, v_1), \diamondsuit_{(0,\delta]}(\text{value}(x, v_2), v_1 > v_2)$ where δ is a "small enough" positive real number $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$
$C_1 : \mathit{trend}(\mathsf{down})$	$\tau_C(f_C(x)) \leftarrow \tau_{C_1}(x), \neg\, \text{notTrendDown}_{C_1}(x)$ $\text{notTrendDown}_{C_1}(x) \leftarrow \text{value}(x, v_1), \diamondsuit_{(0,\delta]}(\text{value}(x, v_2), v_1 < v_2)$ where δ is a "small enough" positive real number $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v).$

Boolean D	Encoding of D
$D = C$	$p_D \leftarrow \diamondsuit_{[0,\infty)} \tau_C(x).$ $\quad$ $p_D \leftarrow \diamondsuit_{[0,\infty)} \tau_C(x).$
$D = D_1$ and D_2	$p_D \leftarrow p_{D_1}, p_{D_2}.$
$D =$ not D_1	$p_D \leftarrow \neg p_{D_1}.$
$\mathit{message}(m) = D$	$\mathit{message}(m) \leftarrow p_D.$

Table 2. The encoding *SDRL* language into extended *DatalogMTL*. Each signal processing expression in the left column, the corresponding *DatalogMTL* rules are provided in the right.

$\tau_C(x) \leftarrow Q(x), \text{value}(x, v)$ we have that $\mathfrak{M}, t \models \tau_C(o_s)$. Since, $\text{value}(x, v)@t \in \mathcal{D}_{\mathcal{S},\mathcal{K}}$ we also have that $\mathfrak{M}, t \models \text{value}(o_s, v)$.

Assume now that $C = \{s_i, \ldots, s_m\}$ and $s = s_i$ for some i. Then it must be $s^{\mathcal{I}} \in C^{\mathcal{I}}$. Next, let us assume that $f_s(t) = v$. Since s is a basic concept, we have that $\text{value}(o_s, v)@t$ is in $\mathcal{D}_{\mathcal{S},\mathcal{K}}$, and thus $\mathfrak{M}, t \models \text{value}(o_s, v)$. Further, following the encoding rule for C, $\tau_C(x) \leftarrow \text{value}(x, v)$, we have that $\mathfrak{M}, t \models \tau_C(o_s)$.

Induction Step: Consider now that C is an expression that is created by other expressions in at most $n+1$. For example, let us assume that $C \leftarrow C_1 : duration(\geq, t')$. Induction step for the other rules can be shown analogously.

We assume that $s^{\mathcal{I}} \in C^{\mathcal{I}}$ and $f_s(t) = v$ for some t and v. Since C is created from C_1 then it must exists s_1 such that $o_s = f_C(o_{s_1})$ and $f_{s_1}(t) = v$ for some interval I that contains t and is longer t'. Since, C_1 is created in at most n steps by induction hypothesis we have that $\mathfrak{M}, t \models \tau_{C_1}(o_{s_1})$ and $\mathfrak{M}, t \models \text{value}(o_{s_1}, v)$. Now we analyze the encoding rule $\tau_C(f_C(x)) \leftarrow \diamondminus_{[0,t']} \boxplus_{[0,t']} \tau_{C_1}(x)$. Intuitively, the body of the rule evaluates to true for some x if there exists a time point in the "past" of t (expressed with condition $\diamondminus_{[0,t']}$) contained in an interval of size t' (expressed with condition $\diamondminus_{[0,t']}$) such that on that interval $\tau_{C_1}(x)$ is true, i.e, $\mathfrak{M}, t'' \models \tau_{C_1}(x)$ for all $t'' \in I$. Since I is such interval for which $\tau_{C_1}(o_{s_1})$ is true, we have that encoding rule fires and makes $\tau_C(f_S(o_s))$ true at point t, i.e., $\mathfrak{M}, t \models \tau_C(f_S(o_{s_1}))$. Furthermore, from the rule $\text{value}(f_C(x), v) \leftarrow \tau_C(f_C(x)), \text{value}(x, v)$ and the fact that $\mathfrak{M}, t \models \tau_C(f_S(o_{s_1})), \text{value}(s_1, v)$ it holds that $\mathfrak{M}, t \models \text{value}(f_C(o_{s_1}), v)$. This concludes the proof.

The following lemma defines a correspondence between a Boolean combinations of signal processing expressions and their encoding rules.

Lemma 2. *Let $\Pi = (\mathcal{S}, \mathcal{K}, \mathcal{M})$ be an SDRL program and $\Sigma_\Pi = (\mathcal{D}_{\mathcal{S},\mathcal{K}}, \Pi_{\mathcal{M}})$ be the extended Datalog$_{nr}$MTL program as defined above. Further, let $\mathcal{I}$ be the canonical interpretation for Π and $\mathfrak{M}$ be the canonical interpretation for Σ_Π. Then, for a Boolean combination of signal processing expression D we have that the following is equivalent:*

$$D^{\mathcal{I}} \text{ is true} \quad \text{iff} \quad \mathfrak{M}, t \models p_D \text{ for any time point } t$$

Proof. The proof is based on induction on the size of D.

Induction Base: We assume $D = C$ for complex expression C and assume $D^{\mathcal{I}}$ is true. Then there must exists a signal s such $o_s \in C^{\mathcal{I}}$ which has at least one value v at some time point t. From Lemma 1 we have that $\mathfrak{M}, t \models \tau_C(o_s)$. Thus, from the encoding rule $p_D \leftarrow \tau_C(x)$ we have that $\mathfrak{M}, t \models p_D$.

Induction Step: We prove induction step for the case $D = D_1$ and D_2. Similarly, it can be shown in case $D = \neg D_1$. Assume that D is true in $\mathcal{I}$, then also D_1 and D_2 are true. Since D_1 and D_2 are constructed in less steps then D by induction hypothesis we have that $\mathfrak{M}, t \models p_{D_1}$ and $\mathfrak{M}, t \models p_{D_2}$. Hence, $\mathfrak{M}, t \models p_D$.

For an extended Datalog*MTL* program Σ, ground atom A and a time point t we define that $\Sigma \models A@t$ for the canonical model $\mathfrak{M}$ of Σ it holds $\mathfrak{M}, t \models A$ Then, directly from Lemmas 1 and 2 we have the following theorem.

Theorem 1 (Encoding Theorem). *Let Π be a program and r a message rule. Let Σ_Π the extended DatalogMTL that encodes Π as described above and let the grounded propositional m_r be the head of DatalogMTL rule encoding r. Then:*

$$\Pi \text{ fires } r \quad \text{iff} \quad \Sigma_\Pi \models m_r@t, \text{ for any time point } t$$

It is not hard to show the ideas of Theorem 5 in [7] can be carried to the extended *Datalog$_{nr}$MTL* programs and thus preserve computational properties required by OBDA. Formally, that means that *data complexity* [5] for the fact-entailment problem is in *AC*^{0}.

The second observation is that we can extend rewriting techniques developed for *Datalog$_{nr}$MTL* in [7] that allow us to rewrite our rules into standard SQL. More involving part of rewriting lies on rewriting algebra of intervals, and for more details we refer [7]. Rewriting that includes functional symbols, negation, aggregation, and built-in arithmetic can be done straightforwardly.

Let Σ be an extended *Datalog$_{nr}$MTL* program, $\mathcal{D}$ a set of facts and A an grounded atom. As usual, $\Sigma, \mathcal{D} \models A@t$ for some time point t holds if for the canonical model $\mathfrak{M}$ of $\Sigma \cup \mathcal{D}$ it holds $\mathfrak{M}, t \models A$. The decision problem *success* is the problem of checking whether $\Sigma, \mathcal{D} \models A@t$. We refer to program (resp. data) complexity if all parameters are fixed except the program (resp. set of facts).

Lemma 3. *Success problem for extended Datalog$_{nr}$MTL programs is PSPACE-complete in combined and program complexity and in AC^0 in data complexity.*

Proof (Proof Idea). Hardness follows from Theorem 5 in [7]. To show membership it is sufficient to observe that each derivation in an extended *Datalog$_{nr}$MTL* program is of length polynomial in the size of the program. Thus it is in *PSPACE*.

From Lemma 3 and Theorem 1 we have the following.

Theorem 2. *The problem of checking whether a message rule is fired is PSPACE-complete (it is complete already in size of rule signal expressions), and it is in AC^0 in the size of signal data and ontological data.*

6 Related Work

In the Semantic Web community it is common to use rule-based ontological languages such as SWRL [10], OWL 2 RL [11], SPIN [12], and SHACL Rules [13–15] for analytics. Compared with *SDRL*, these languages cannot be directly used in the context of signal analysis as they do not support reasoning of values over the temporal dimension.

Works in [16, 17] introduce analytical operations directly into ontological rules in such a way that OBDA scenario is preserved. They define analytical functions on concepts, e.g. avg C, in OBDA setting. However, the authors do not consider temporal dimension of the rules.

As discussed above, our work is strongly related to the work on well-studied Metric Temporal Logic [18]. In particular, we use a non-trivial extension of non-recursive Datalog language Datalog$_{\text{nr}}$*MTL* which is suitable for OBDA scenario.

Another related direction is real-time processing of signal data streams. In this direction, most of the work done so far mainly focused on querying RDF stream data. Many different approaches such as C-SPARQL [19], SPARQL$_{stream}$ [20] and CEQLS [21] have surfaced in recent years, introducing SPARQL based query processors. Most of them, apart from C-SPARQL, follow the Data Stream Management Systems (DSMSs) paradigm and do not provide support for stream reasoning. EP-SPARQL [22] combines SPARQL with complex event processing features, and includes sequencing and simultaneity operators. Unlike the others, LARS [23] is an Answer Set Programming based framework, which enables reasoning by compiling a Knowledge Base together with a SPARQL-like query into a more expressive logic program.

Acknowledgments This research is supported by the EPSRC projects MaSI^3, DBOnto, ED^3; and by the Free University of Bolzano projects QUEST, OBATS and ROBAST.

References

1. G. Vachtsevanos, F. L. Lewis, M. Roemer, A. Hess, B. Wu, Intelligent Fault Diagnosis and Prognosis for Engineering Systems, Wiley, 2006.
2. E. Kharlamov, O. Savkovic, G. Xiao, R. Penaloza, G. Mehdi, I. Horrocks, M. Roshchin, Semantic rules for machine diagnostics: Execution and management, in: CIKM, 2017.
3. G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalaycı, S. Brandt, I. Horrocks, M. Roshchin, T. A. Runkler, Semantic rule-based equipment diagnostics, in: ISWC, 2017, pp. 314–333.
4. G. Mehdi, E. Kharlamov, O. Savkovic, G. Xiao, E. G. Kalaycı, S. Brandt, I. Horrocks, M. Roshchin, T. Runkler, Semantic rules for siemens turbines, in: ISWC (Posters and Demos), 2017.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, JAR 39 (3).
6. E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3).
7. S. Brandt, E. G. Kalaycı, R. Kontchakov, V. Ryzhikov, G. Xiao, M. Zakharyaschev, Ontology-based data access with a horn fragment of metric temporal logic, in: AAAI, 2017.
8. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semantics 10 (2008) 133–173.
9. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, New York, NY, USA, 2003.
10. I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and RuleML, W3C Member Submission, World Wide Web Consortium (2004).
11. B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, Owl 2 web ontology language: Profiles, W3C Recommendation, World Wide Web Consortium, http://www.w3.org/TR/owl2-profiles/ (2012).
12. H. Knublauch, J. A. Hendler, K. Idehen, Spin - overview and motivation, W3C member submission (February 2011).
13. W3C, SHACL advanced features, W3C working group note, W3C (June 2017).
14. J. Corman, J. L. Reutter, O. Savkovic, Semantics and validation of recursive SHACL, in: ISWC, 2018.
15. J. Corman, J. L. Reutter, O. Savkovic, Tractable notion of stratification for SHACL, in: ISWC, 2018.
16. E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, P. Haase, I. Horrocks, H. Kllapi, M. Koubarakis, Ö. L. Özçep, M. Rodriguez-Muro, R. Rosati, M. Schmidt, R. Schlatte, A. Soylu, A. Waaler, Optique: Towards OBDA systems for industry, in: ESWC Satellite Events, 2013.
17. E. Kharlamov, N. Solomakhina, Ö. L. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, S. Watson, How semantic technologies can enhance data access at siemens energy, in: ISWC, 2014, pp. 601–619.
18. R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (4).
19. D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, M. Grossniklaus, C-SPARQL: a continuous query language for RDF data streams, Int. J. Semantic Computing 4 (1) (2010) 3–25.
20. J. Calbimonte, H. Jeung, Ó. Corcho, K. Aberer, Enabling query technologies for the semantic sensor web, Int. J. Semantic Web Inf. Syst. 8 (1) (2012) 43–63.
21. D. L. Phuoc, M. Dao-Tran, J. X. Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: ISWC, 2011, pp. 370–388.

22. D. Anicic, P. Fodor, S. Rudolph, N. Stojanovic, EP-SPARQL: a unified language for event processing and stream reasoning, in: WWW, 2011, pp. 635–644.
23. H. Beck, M. Dao-Tran, T. Eiter, M. Fink, LARS: A logic-based framework for analyzing reasoning over streams, in: AAAI, 2015, pp. 1431–1438.