

A Non-Deterministic Multiset Query Language

Bartosz Zieliński*

Department of Computer Science

Faculty of Physics and Applied Informatics, University of Łódź

Pomorska 149/153 90-236 Łódź, Poland

bartosz.zielinski@fis.uni.lodz.pl

Abstract. We develop a multiset query and update language executable in a term rewriting system. Its most remarkable feature, besides non-standard approach to quantification and introduction of fresh values, is non-determinism — a query result is not uniquely determined by the database. We argue that this feature is very useful, e.g., in modelling user choices during simulation or reachability analysis of a data-centric business process — the intended application of our work. Query evaluation is implemented by converting the query into a terminating term rewriting system and normalizing the initial term which encapsulates the current database. A normal form encapsulates a query result. We prove that our language can express any relational algebra query. Finally, we present a simple business process specification framework (and an example specification). Both syntax and semantics of our query language is implemented in Maude.

Keywords: term rewriting, query languages, business process modelling

1. Introduction

In a data-centric approach to business process modelling (see, e.g., [1, 2]), specification of data transformation during case execution is an integral part of the business process model. This new paradigm requires new tools and formalisms for effective specification, simulation and validation. Task-centric

*Address for correspondence: Department of Computer Science, Faculty of Physics and Applied Informatics, University of Łódź, Pomorska 149/153 90-236 Łódź, Poland.

models are commonly formalized using Petri nets (see, e.g., [3, 4]). Adapting Petri net-based formalizations to data-centric models is, however, problematic: While simple transformations on data *can* be represented directly within a coloured Petri net, Petri nets lack facilities for complex data processing and querying. Even so, there exists a large amount of literature (see e.g., [5, 6, 7]) devoted to enriching Petri nets with data processing capabilities and automated verification of their properties. Recent paper [8] introduced *DB-Nets* — an attempt to integrate coloured Petri nets with relational databases. The use of two separate formalisms complicates verification and simulation (though it corresponds to actual implementations of BPM systems). In the following paper [9] a subset of database operations was implemented inside coloured Petri nets with name creation and transition priorities.

Conditional term rewriting [10] was proposed as an alternative (if less popular) generic framework for specification of dynamic systems [11]. It subsumes a variety of Petri nets [12] and their simulation is one of popular applications of the term rewriting system Maude [13, 14]). More precisely, it is well known (see e.g., [12]) that coloured Petri nets can be implemented by multiset rewriting systems, and, conversely, rewriting systems which rewrite multisets of terms representing colour tokens can be interpreted as Petri nets: just identify places with colour tokens, and each rewriting rule of the form

$$a_1 a_2 \dots a_n \Rightarrow b_1 b_2 \dots b_m$$

with a transition with input arcs from a_1, a_2, \dots, a_n and output arcs to b_1, b_2, \dots, b_m . Other constructs, such as inhibitor arcs can be easily implemented with conditional rewriting rules. Rewriting systems are more general than Petri Nets since they are not limited to rewriting multisets (on the other hand, Petri nets are much better supported by tools and programming libraries). However, rewriting systems still share with Petri nets the limitation and inconvenience of not directly supporting bulk, complex operations on data which involve some kinds of quantification. For example, suppose that the state of a data driven business process related to e-commerce is represented by a multiset of terms. In particular, terms of the form $\text{item}(p, c)$ denote the presence of a product p in the basket of a customer c . Suppose now that c cancels the case, and so we need to remove all c 's items from the multiset. Describing removal of a single (nondeterministically chosen) item is easy with the rule $\text{item}(c, x) \Rightarrow \emptyset$ where x is a variable, but specifying in a rewriting system (or a Petri net) that all of them need to be removed before the next business step is more complex (though clearly possible) and would require auxiliary tokens and conditional rewrite rules corresponding to inhibitor arcs in a Petri net preventing other transitions as long as there are still some c 's items present in the rewritten multiset. Thus, a high-level query language adding quantifier constructs on top of conventional rewriting system or a Petri net is clearly desirable, particularly if rewriting systems or Petri nets are to become convenient formalisms to model data driven business processes.

In this paper we present an expressive multiset query and update language $\mathcal{Q}_{\Sigma, \mathcal{D}}$, designed to be executable in a term rewriting system, useful for a unified and somewhat “Petri netty” formalization of data-centric business processes. The connection with Petri nets is admittedly tenuous and follows from the fact that the language acts on data represented as multisets of terms which could be viewed as tokens (see the discussion above). Since this language specifies changes to data, instead of being a reimplement of relational calculus, it contains linear-like features fitting a term rewriting implementation. Most remarkably, $\mathcal{Q}_{\Sigma, \mathcal{D}}$ is non-deterministic — the result of a query or update is not,

in general, uniquely determined by the database. This permits modelling user choices, just like in the case of Petri nets. $\mathcal{Q}_{\Sigma, \mathcal{D}}$ consists of three sublanguages, parametrized with respect to a signature Σ and Σ -algebra of facts \mathcal{D} :

1. Language $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ of conditions (Boolean queries) which can be used independently as constraints, or as components of queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ and $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$.
2. Data manipulation language $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$. A DML query Q in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ defines new facts to be added to the database. Some of the old facts used in constructing the new ones may be deleted.
3. Language $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ of queries which only return facts but do not change the database. Both syntax, and to some extent semantics of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ is a restriction of syntax and semantics of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$.

A query Q in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\alpha}$, $\alpha \in \{\text{cnd}, \text{qry}, \text{dml}\}$, is given semantics by assignment of a rewriting system $\mathcal{R}_{\Sigma, \mathcal{D}}^{\alpha}(Q)$. To evaluate Q in a database F we start with an initial term $I_Q(F)$. A normal form of $I_Q(F)$ wraps a result of Q 's evaluation: a Boolean value indicating validity of a condition, a query answer or a new database resulting from execution of a DML query. As remarked above, while $\mathcal{R}_{\Sigma, \mathcal{D}}^{\alpha}(Q)$ is always terminating (i.e., there are no infinite execution paths, so we always do get *some* result from evaluating Q), it is not confluent (i.e., divergent execution paths may not eventually converge) in general, hence we may get distinct results depending on nondeterministic choices. We identify syntactic constraints on queries in each of the sublanguages which ensure confluence for the rewriting system, and hence determinism for the results of the query. $\mathcal{Q}_{\Sigma, \mathcal{D}}$ shares with the language introduced in [15] a non-standard approach to variable binding. The approach avoids problems with capture-free substitutions without dispensing with explicit variables, but at the price of non-compositionality: The surrounding context may determine whether a variable in a subterm is free or bound *in this subterm*. $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ supports introduction of fresh values to the database, which is used, e.g., to generate identifiers for newly created artifacts or to simulate user input (cf. [7]).

Since queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}$ are converted to a term rewriting system, verification of a business process specified as a set of DML queries (see Section 7) can be assisted with symbolic reachability analysis techniques based on narrowing (see e.g., [16]). We plan to expand on this idea in future research. Note that our results in this article regarding the confluence of the rewriting systems to which a particular subclass of queries compiles to, may be relevant for narrowing (see e.g., [17, 18], c.f. [19]). E.g., narrowing a confluent system may provide a more efficient search procedure than in the case of a non-confluent one.

1.1. Prior work

The present paper builds on the previous paper [15] (cf. [20]) where a multiset query language executed in Maude was proposed. $\mathcal{Q}_{\Sigma, \mathcal{D}}$ shares many similarities with the language described in [15], particularly the treatment of quantification. It has, however, distinct syntax (with, e.g., fact markings in quantifiers) and distinct semantics. We consider both languages to be alternatives, each of which with its own strengths. A side-by-side comparison is presented in Table 1. Observe that while the language described in [15] can be defined both in the set and multiset setting (in the former case we match multisets of facts modulo idempotence in addition to commutativity, associativity and identity), here

we assume exclusively multiset setting. This is because the language described here is implemented through multiset rewriting. Making multiset constructor idempotent would make it impossible to consistently replace matched terms, a feature which is crucial for our formalism to behave sensibly. E.g., given a rule $a \Rightarrow b$, term $ab =_{\mathcal{A}} aab$ rewrites in one step both to $bb =_{\mathcal{A}} b$ (intended) and $abb =_{\mathcal{A}} ab$ (not intended).

Table 1. Comparison between $\mathcal{Q}_{\Sigma, \mathcal{D}}$ and $\mathcal{CF}_{\Sigma, \mathcal{D}}(X)$ from [15]

$\mathcal{Q}_{\Sigma, \mathcal{D}}$	$\mathcal{CF}_{\Sigma, \mathcal{D}}(X)$
Semantics of (possibly) non-deterministic queries is based on translation into term rewriting systems.	Semantics of queries is based on term matching on the metalevel. Queries are always deterministic.
Queries return facts in the same signature as the database against which the query is evaluated.	Queries return objects of an arbitrary signature (as long as it contains a “union” operator).
DML queries construct multisets of facts to be added to the current database. Some of the current facts used in the construction may be deleted.	DML expressions construct a pair of sets or multisets of facts — those to be deleted from and those to be added to the current database.
Fresh values are introduced through “virtual fresh facts”. Actual freshness is ensured through rewrite rules which define the semantics of DML query.	Attributes of input facts can be marked by special sorts as fresh. Testing framework ensures that values injected in those columns are actually fresh.
It is assumed that the database is a <i>multiset</i> of facts	The language can be defined both in set and multiset setting
Treatment of variable binding is identical in both languages	

$\mathcal{Q}_{\Sigma, \mathcal{D}}$, has some similarities to matching logic [21, 22], as both are based on term matching. They have different purposes, however, and different syntax and semantics. Unlike matching logic statements, $\mathcal{Q}_{\Sigma, \mathcal{D}}$ queries are non-deterministic. Matching logic is used for software verification, while $\mathcal{Q}_{\Sigma, \mathcal{D}}$ is a query language intended to be a *component* of the system. Finally, matching logic has conventional quantifiers, whereas we use a non-standard quantification over “relation patterns”.

CINNI [23] is a generic calculus of substitutions implemented in Maude which combines de Bruijn indices with explicit names to solve the problem of capture-free substitutions. To avoid the associated complexity, we decided not to use conventional variable binding implemented, e.g., using CINNI.

Our quantification over “relation patterns” instead of variables, resembles quantifier constructs in description logic¹ [24]. Our syntactic construct, however, uses explicit variable names and is not limited to binary relations where only the second column is bound by the quantifier.

Data-centric business process models are formalized in a variety of ways. First-order logic and its restricted variations (see e.g., [25, 26, 27, 28]), datalog [29], and UML [30], are popular choices. Those formalisms are excellent for the specification of data models, and they come with expressive query languages; they are, however, not so ideally suited for modelling change, because of frame problems [31], where it is not always obvious what information is modified and what stays the same. Rewriting formalisms [10], which are explicit about scope of change have a clear advantage here. On the other hand, a great deal of work has been devoted to formal verification of logic based business

¹We are grateful to Prof. Andrzej Tarlecki for this observation.

process formalism, see e.g., [32, 33] in the context of hierarchical artifact systems. For another example see [34] where a data aware extension of BPMN was proposed together with SMT (satisfiability modulo theories) based verification techniques.

In [35] a language called Reseda, was introduced for specification of data driven business process. The language integrates data description with behaviour. What makes it relevant as a prior work to the present paper is that Reseda's semantics is defined by associating with a Reseda program a transition system. Such a program can then be executed by rewriting data in accordance with the transition rules, similarly to the execution of the language described here.

As we remarked earlier, a recent paper [8] introduced *DB-Nets* which integrate coloured Petri nets with relational databases. Since the use of two separate formalisms complicates verification and simulation in the following paper [9] a subset of database operations was implemented inside coloured Petri nets with name creation and transition priorities. Thus, the motivation of [9] is analogous to the motivation of this paper, but in the world of Petri nets instead of term rewriting systems. There are however two important differences: First, our language is meant to provide complete specification of data driven business processes, whereas in [9] the business process is still specified as a Petri net, and there is just an interface between relational queries (perhaps implemented inside the net itself) and the main net describing the process. Secondly (and this is what makes the first point possible) we do not simply implement a conventional relational dml and query language in a rewriting system. Instead, we implement a *linear* and *non-deterministic* query language which can emulate user choices and creation of new objects.

1.2. Preliminaries on term rewriting

We recall basic notions related to term rewriting [10, 11], and many sorted equational logic [36].

Let S be a poset (partially ordered set). A family $X = \{X_s \mid s \in S\}$ of sets is called *an S -sorted set* if $X_s \subseteq X_{s'}$ whenever $s \leq s'$. We abbreviate $x \in \bigcup X$ as $x \in X$. We write $x : s$ iff $x \in X_s$.

An *algebraic signature* $\Sigma = (\Sigma_S, \Sigma_F)$ consists of a finite poset of sorts Σ_S and a finite set Σ_F of function symbols. The set of function symbols Σ_F is Σ_S^+ -sorted, where Σ_S^+ is the set of finite, non-empty sequences of elements of Σ_S partially ordered with $s_0 \cdots s_n \leq t_0 \cdots t_m$ iff $m = n$, $s_0 \leq t_0$ and $s_i \geq t_i$ for all $i \in \{1, \dots, n\}$. Traditionally we write $f : s_1 \dots s_n \rightarrow s_0$ when $f \in (\Sigma_F)_{s_0 \dots s_n}$, where we denote by $(\Sigma_F)_{s_0 \dots s_n}$ the set of function symbols of sort $s_0 \dots s_n$. This explains the somewhat confusing ordering on Σ_S^+ : we are covariant on return value and contravariant on arguments. Symbols $c := s$ are called *constants* of sort s . A Σ -algebra \mathbb{A} is an assignment of a set $\llbracket s \rrbracket_{\mathbb{A}}$ to each $s \in \Sigma_S$ such that $\llbracket s \rrbracket_{\mathbb{A}} \subseteq \llbracket s' \rrbracket_{\mathbb{A}}$ if $s \leq s'$, and a function $\llbracket f \rrbracket_{\mathbb{A}} : \llbracket s_1 \rrbracket_{\mathbb{A}} \times \cdots \times \llbracket s_n \rrbracket_{\mathbb{A}} \rightarrow \llbracket s \rrbracket_{\mathbb{A}}$ to each $f : s_1 \dots s_n \rightarrow s$ in Σ_F . Let $V := \{V_s \mid s \in \Sigma_S\}$ be a Σ_S -sorted set of variables. A term algebra $\mathcal{T}_{\Sigma}(V)$ has “sort-safe” terms as elements and function symbols interpreted by themselves. We denote by \mathcal{T}_{Σ} the algebra of ground Σ -terms. We often use mixfix syntax where underscores in the function name correspond to consecutive arguments. Thus, if Σ_F contains $_ + _ : A A \rightarrow A$ and $0 := \rightarrow A$ then $0 + 0$ is a ground term of sort A . Positions in a term are denoted by strings of positive integers. Denote by ε the empty string, and by $t|_{\kappa}$ the subterm of $t \in \mathcal{T}_{\Sigma}(V)$ at position $\kappa \in \mathbb{Z}_+^*$ (if defined), i.e., $t|_{\varepsilon} := t$, and $f(t_1, \dots, t_n)|_{k\kappa} := t_k|_{\kappa}$. Let $Pos(t) := \{\kappa \in \mathbb{Z}_+^* \mid t|_{\kappa} \text{ is defined}\}$. If $\kappa \in Pos(t)$ and u is a term of the same sort as $t|_{\kappa}$, then we denote by $t[u]_{\kappa}$ the result of replacing $t|_{\kappa}$ in t with u .

We use a standard notation for substitutions. Let $\vec{a} = a_1, \dots, a_n$ be a list of terms, $\vec{v} = v_1, \dots, v_n$ a list of distinct variables. Then we denote $\sigma = \{\vec{a}/\vec{v}\} = \{a_1/v_1, \dots, a_n/v_n\}$ when $\sigma(v_i) = a_i$, $i \in \{1, \dots, n\}$, and $\sigma(v) = v$ for any variable $v \notin \{v_1, \dots, v_n\}$.

A Σ -algebra may be defined as a quotient of \mathcal{T}_Σ by a congruence generated by a set $A \cup E$ of equalities, where equalities in A , referred to as *equational attributes*, define structural properties such as associativity, commutativity, or identity, and E consists of conditional equalities interpreted as directed simplification rules on \mathcal{T}_Σ . It is assumed that simplifications terminate and are confluent, hence each t has the unique (modulo A) irreducible form $t \downarrow_{E/A} \in \mathcal{T}_\Sigma$ representing a class of t in $\mathcal{T}_\Sigma / \equiv_{A \cup E}$.

Simplification with respect to equalities computes values. The behaviour is represented with rewritings. A rewriting system $\mathcal{R} = (\Sigma, A, E, R)$ consists of a signature Σ , a set of equations $A \cup E$ where E defines confluent and terminating (modulo A) simplifications on \mathcal{T}_Σ , and a finite set R of conditional rewriting rules of the form $\lambda : t_1 \Rightarrow t_2$ if C , where optional condition C is a conjunction of equalities, and λ is the rule's label. A one-step rewrite $u \xrightarrow{\lambda}_{\mathcal{R}} u'$ from u to u' using such a rule is possible if there exists a position κ , term v , and a substitution σ such that $u =_A v$, $v|_{\kappa} =_A \sigma(t_1)$, $u' =_A v[\sigma(t_2)]_{\kappa}$ and $\sigma(C)$ is satisfied. We write $u \rightarrow_{\mathcal{R}} u'$ iff there exist terms $s, s' \in \mathcal{T}_\Sigma$ and a label λ of a rule in R such that $u \downarrow_{E/A} =_A s$, $s \xrightarrow{\lambda}_{\mathcal{R}} s'$, and $s' \downarrow_{E/A} =_A u' \downarrow_{E/A}$. We denote by $\rightarrow_{\mathcal{R}}^+$ and $\rightarrow_{\mathcal{R}}^*$ the transitive and reflexive-transitive closures of $\rightarrow_{\mathcal{R}}$. We also write $u \rightarrow_{\mathcal{R}}^! u'$ if $u \rightarrow_{\mathcal{R}}^* u'$ and there is no u'' such that $u' \rightarrow_{\mathcal{R}} u''$. If \mathcal{R} is implied by the context, we omit \mathcal{R} from arrows.

Variants of the following definition and easy to prove lemma appear in the literature (see, e.g., [37]):

Definition 1.1. Let (X, \rightarrow) , where $\rightarrow \subseteq X \times X$, be a transition system. Assume that \equiv is an equivalence on X which is a bisimulation on (X, \rightarrow) . We call \rightarrow *semiconfluent at $x \in X$ modulo \equiv* if for all $y, y' \in X$ such that $y \leftarrow x \rightarrow^* y'$ there exist $z, z' \in X$ such that $y \rightarrow^* z \equiv z' \leftarrow y'$. We call \rightarrow *semiconfluent modulo \equiv* if \rightarrow is semiconfluent at all $x \in X$. We call \rightarrow *confluent at $x \in X$ modulo \equiv* if for all $y, y' \in X$ such that $y \leftarrow^* x \rightarrow^* y'$ there exist $z, z' \in X$ such that $y \rightarrow^* z \equiv z' \leftarrow^* y'$. We call \rightarrow *confluent modulo \equiv* if \rightarrow is confluent at all $x \in X$.

Lemma 1.2. Semiconfluence modulo equivalence implies confluence modulo equivalence.

2. Multisets of facts, fresh facts and patterns

Our queries are evaluated against, or act on, finite multisets of facts. Duplicate facts can be genuinely useful and removing them is computationally expensive. If necessary, duplicates can be removed explicitly or, better, one can ensure that no duplicates are introduced in the first place by judicious choice of DML operations. In fact, SQL is a multiset query language as well, hence by using multisets we are closer to the actual relational database practice than formal systems based on sets.

$\mathcal{Q}_{\Sigma, \mathcal{D}}$ is parametrized with respect to a signature of facts Σ and a Σ -algebra of facts \mathcal{D} . $\Sigma_{\mathcal{F}}$ must contain sorts **Fact** and **Bool** for facts and Booleans, respectively. All constructors for facts are contained in $\Sigma_{\mathcal{F}}$. Facts are reifications of predicate instances. A typical fact has the form $f(a_1, \dots, a_n)$, where $f : s_1 \dots s_n \rightarrow \mathbf{Fact}$ is a fact constructor. \mathcal{D} defines all the data types used in facts and is

specifiable in terms of directed equations and equational attributes. \mathcal{D} must define Boolean connectives and Boolean-valued equality $_ = _ : s \rightarrow \text{Bool}$ for all $s \in \Sigma_{S,K}$. We assume that all ground terms of sort Bool simplify to either \mathbf{t} or \mathbf{f} . This is non-trivial: Define a function $f : \text{Nat} \rightarrow \text{Bool}$ with a single equation $f(0) = \mathbf{t}$. Then $f(1)$ is fully reduced and distinct from both \mathbf{t} and \mathbf{f} .

Multisets of facts. The signature of multisets extends Σ_S with sorts $(\text{Ne})\text{FSet}$ of finite (non-empty) multisets of facts. The subsort ordering is given by $\text{Fact} < \text{NeFSet} < \text{FSet}$. In particular, each fact is a non-empty multiset of facts. Finite multisets of facts are constructed with an associative and commutative binary operator $_ \circ _ : \text{FSet} \text{FSet} \rightarrow \text{FSet}$ (cf. [38]) with identity element $\emptyset : \rightarrow \text{FSet}$. Operator $_ \circ _$ is subsort overloaded with the additional declaration $_ \circ _ : \text{FSet} \text{NeFSet} \rightarrow \text{NeFSet}$. Thus, a multiset constructed from a multiset and a non-empty multiset is non-empty.

Freshness and nominal sorts. Support for creation of fresh values is a common requirement (cf. [7]): Identifiers for new objects must not belong to the present nor any past active domain of the database. To understand why reusing identifiers from past domains is bad consider situation where a new business object is created with the identifier of a previously deleted one. In this case the attempt to verify if the deleted object is present in the final database may yield an incorrect affirmative answer. We support creation of fresh values of *nominal sorts* only. Usually this suffices, and freshness for non-nominal data types is problematic (cf. [39]). A sort s is nominal (relative to Σ -algebra \mathcal{D}) if values of this sort have no non-trivial algebraic or relational structure beside equality. In particular, for nominal s , $s' \leq s \leq s''$ if and only if $s' = s = s''$. To create values of each nominal sort s we have constructor $i_-^s : \text{Nat} \rightarrow s$ which belongs *neither* to Σ *nor* to the signature of $\mathcal{Q}_{\Sigma, \mathcal{D}}$.

Example 2.1. Consider a client basket database. Identifiers of customers, products and baskets have sorts \mathbf{c} , \mathbf{p} , and \mathbf{b} , respectively. We use two fact constructors: $_ \text{owns} _ : \mathbf{c} \ \mathbf{b} \rightarrow \text{Fact}$ and $_ \text{in} _ : \mathbf{p} \ \mathbf{b} \rightarrow \text{Fact}$. Multiset of facts $(i_1^{\mathbf{c}} \text{ owns } i_1^{\mathbf{b}}) \circ (i_2^{\mathbf{p}} \text{ in } i_1^{\mathbf{b}}) \circ (i_3^{\mathbf{p}} \text{ in } i_1^{\mathbf{b}})$ denotes the state in which customer $i_1^{\mathbf{c}}$ is the owner of basket $i_1^{\mathbf{b}}$ containing products $i_2^{\mathbf{p}}$ and $i_3^{\mathbf{p}}$. Using multisets instead of sets can be useful: multiset $(i_3^{\mathbf{p}} \text{ in } i_1^{\mathbf{b}}) \circ (i_3^{\mathbf{p}} \text{ in } i_1^{\mathbf{b}})$ denotes the situation where basket $i_1^{\mathbf{b}}$ contains two items of $i_3^{\mathbf{p}}$.

Constructing values of nominal sorts from natural numbers simplifies creation of fresh values: to ensure freshness one can construct the new value with the smallest natural number which was not used so far. To keep track of those “smallest unused naturals”, and to make retrieval of fresh values similar to retrieval of data we use fresh facts of sort $\text{Fact}^{\mathbf{n}}$ unrelated to Fact . For each nominal sort s we have a single-argument constructor of the form $C_s : s \rightarrow \text{Fact}^{\mathbf{n}}$ which wraps value i_n^s such that for all $m \geq n$, i_m^s was never used before. When fresh value of sort s is requested, we return i_n^s and update the fresh fact to $C_s(i_{n+1}^s)$. Fresh facts are combined into (non-empty) multisets of sort $(\text{Ne})\text{FSet}^{\mathbf{n}}$ using commutative and associative operator $_ \circ _ : \text{FSet}^{\mathbf{n}} \text{FSet}^{\mathbf{n}} \rightarrow \text{FSet}^{\mathbf{n}}$ with identity $\emptyset : \rightarrow \text{FSet}^{\mathbf{n}}$. To facilitate bulk updates of fresh facts needed in the semantics of DML queries, we define the following function:

$$v : \text{FSet}^{\mathbf{n}} \rightarrow \text{FSet}^{\mathbf{n}}, \quad v(C_{s_1}(i_{m_1}^{s_1}) \circ \dots \circ C_{s_n}(i_{m_n}^{s_n})) = C_{s_1}(i_{m_1+1}^{s_1}) \circ \dots \circ C_{s_n}(i_{m_n+1}^{s_n}). \quad (1)$$

Patterns. Quantifiers in $\mathcal{Q}_{\Sigma, \mathcal{D}}$ quantify over patterns (of sort Pat) containing non-ground multisets of facts and fresh facts marked by modalities, which control retention of matched facts (i.e., whether upon matching they are removed temporarily, permanently, or not at all from the database),

and syntactically wrap fresh facts (i.e., fresh facts can appear in the pattern only inside specialized modality). Patterns can be preserving (of sort Pat^p), semi-terminating (of sort Pat^{st}), terminating (of sort Pat^t), terminating and preserving (of sort Pat^{tp}), or neither. The subsort relation is defined by $\text{Pat}^{tp} < \text{Pat}^t < \text{Pat}^{st} < \text{Pat}$ and $\text{Pat}^{tp} < \text{Pat}^p < \text{Pat}$. Patterns are constructed with modalities

$$[-]! : \text{NeFSet} \rightarrow \text{Pat}^p, \quad [-]? : \text{NeFSet} \rightarrow \text{Pat}^{tp}, \quad [-]_0 : \text{NeFSet} \rightarrow \text{Pat}^{st}, \quad [-]_n : \text{NeFSet}^n \rightarrow \text{Pat}.$$

and associative and commutative, subsort overloaded operator:

$$\begin{aligned} _ \circ _ : \text{Pat Pat} &\rightarrow \text{Pat}. & _ \circ _ : \text{Pat Pat}^t &\rightarrow \text{Pat}^t, & _ \circ _ : \text{Pat}^p \text{Pat}^p &\rightarrow \text{Pat}^p, \\ _ \circ _ : \text{Pat}^p \text{Pat}^{tp} &\rightarrow \text{Pat}^{tp}, & _ \circ _ : \text{Pat}^{st} \text{Pat} &\rightarrow \text{Pat}^{st}. \end{aligned}$$

Thus, a terminating (resp. a semi-terminating) pattern has to contain at least one fact wrapped with $[-]?$ (resp. with either $[-]?$ or $[-]_0$). A terminating *and* preserving pattern consists of facts marked only with $[-]!$ or $[-]?$, and it contains at least one fact wrapped with $[-]?$. Directed equalities $[F_1]_m \circ [F_2]_m = [F_1 \circ F_2]_m$, where $m \in \{!, ?, 0, n\}$, and F_1, F_2 are non-empty multisets of (fresh) facts guarantee that fully reduced patterns have facts gathered in groups of the same modality.

Example 2.2. Let Id be a sort, let $f, g : \text{Id Nat} \rightarrow \text{Fact}$ and $h : \text{Id} \rightarrow \text{Fact}$ be fact constructors and let x, y, z and t be variables. Then

$$[f(x, y) \circ h(x)]? \circ [g(x, y)]! \circ [C_{\text{Id}}(t)]_n \circ [g(x, 1)]_0$$

is a pattern. It is terminating because of a presence of $[f(x, y) \circ h(x)]?$ subpattern. It is not, however, preserving since it contains facts wrapped in $[-]_0$ and $[-]_n$.

The informal meaning of modalities is as follows: Facts matched by those marked by $[-]?$ can be considered at most once during quantifier evaluation, but they are not removed from the database. Facts marked by $[-]_0$ are removed from the database when matched, but they are returned if the computation branch this matching leads to is unsuccessful. Thus, the presence of facts marked by $[-]_0$ in the pattern may not guarantee termination, unless one can prove that the formula under quantifier is always successful. Facts marked by $[-]!$ are always retained in the database, and $[-]_n$ wraps fresh facts.

Remark 2.3. In what follows we use the following notation. Let P be a pattern. We denote by $P_0, P!, P?, P_n$ the multisets of facts consisting of those facts in P which are wrapped by modalities $[-]_0, [-]!, [-]?$, and $[-]_n$, respectively. Thus, e.g., $([F_1]! \circ [F_2]?)? := F_2$ and $([F_1]! \circ [F_2]?)_n := \emptyset$.

3. Query and condition languages

This section introduces the three sublanguages of $\mathcal{Q}_{\Sigma, \mathcal{D}}$, their syntax and informal semantics. Formal semantics based on conditional term rewriting is provided in the subsequent sections.

3.1. Conditions

The language $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ of conditions on finite multisets of facts is analogous to first-order logic with quantification restricted to the active domain.

Definition 3.1. Let Σ be a signature and let \mathcal{D} be a Σ -algebra of facts. Formulas of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ are (generally non-ground) terms of sort Cnd constructed with

$$\perp : \rightarrow \text{Cnd}, \{ _ \} : \text{Bool} \rightarrow \text{Cnd}, \neg _ : \text{Cnd} \rightarrow \text{Cnd}, _ \vee _ : \text{Cnd} \text{ Cnd} \rightarrow \text{Cnd}, \exists _ _ : \text{Pat}^{\text{tp}} \text{Cnd} \rightarrow \text{Cnd}.$$

Thus, \perp , $\{B\}$, $\neg\psi$, $\psi \vee \psi'$ and $\exists P. \psi$ are conditions if B is a term of sort Bool , P is a terminating and preserving pattern, and ψ and ψ' are conditions. Consider condition $T := \exists P. \psi$. Existential quantifier \exists binds in ψ all the variables appearing in P which were not bound by the term surrounding T . Thus, the meaning of the formula may change when it is placed in a different context.

Example 3.2. Let $R : \text{Nat Nat} \rightarrow \text{Fact}$, and suppose that terms $R(t_1, t_2)$ represent rows of a relation $\mathbf{R} \subseteq \mathbf{N} \times \mathbf{N}$. Let x, y , and z be distinct variables. Then condition $\neg \exists [R(x, y)]?. \exists [R(x, z)]?. \neg \{y = z\}$ expresses functional dependency from the first to the second column of \mathbf{R} . The first quantifier binds x and y , the second one binds z . The condition is closed. The subcondition $\exists [R(x, z)]?. \neg \{y = z\}$ taken on its own is open, but only y is free and the quantifier now binds *both* x and z .

Closed formulas in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ are called *sentences* in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$. Let $\text{Var}(t)$ be the set of variables of t , and let $\text{cl}^?(\phi)$ iff ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ is closed. To define $\text{cl}^?(\phi)$ by structural recursion we need to keep track of variables bound by the context of ϕ . Thus, $\text{cl}^?(\phi) := \text{cl}^?(\phi, \emptyset)$, where, for any set of variables V ,

$$\begin{aligned} \text{cl}^?(\perp, V) &= \mathbf{t}, & \text{cl}^?(\neg \phi, V) &= \text{cl}^?(\phi, V), & \text{cl}^?(\{t\}, V) &= \text{Var}(t) \subseteq V \\ \text{cl}^?(\phi_1 \vee \phi_2, V) &= \text{cl}^?(\phi_1, V) \wedge \text{cl}^?(\phi_2, V), & \text{cl}^?(\exists P. \phi, V) &= \text{cl}^?(\phi, V \cup \text{Var}(P)). \end{aligned} \quad (2)$$

As the syntactic sugar we define operators $\top : \rightarrow \text{Cnd}$, $_ \wedge _ : \text{Cnd} \text{ Cnd} \rightarrow \text{Cnd}$, $\forall _ _ : \text{Pat}^{\text{tp}} \text{Cnd} \rightarrow \text{Cnd}$. with equalities $\top = \neg \perp$, $\phi_1 \wedge \phi_2 = \neg(\neg \phi_1 \vee \neg \phi_2)$, $\forall P. \phi = \neg \exists P. \neg \phi$. Later we prove that, for any condition ϕ , $\neg \neg \phi$ is logically equivalent to ϕ . Then the functional dependency in Example 3.2 can be equivalently written as $\forall [R(x, y)]?. \forall [R(x, z)]?. \{y = z\}$.

Definition 3.3. A subcondition ψ of ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ is a subterm of ϕ of sort Cnd .

3.2. Syntax of queries and DML queries

Syntactically $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ is a restriction of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$. Therefore, we define their syntax jointly as follows:

Definition 3.4. Queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ are terms of sort Qy . DML queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ are terms of sort DQy . Success assured (DML) queries are terms of sorts DQy^s . The sorts are ordered with $\text{Fact} < \text{Qy} < \text{DQy}$ and $\text{Fact} < \text{DQy}^s < \text{DQy}$. Thus, every query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ can be also interpreted as DML query (which

inserts but doesn't delete). Every fact is also a query. Success assured DML queries are DML queries guaranteed to return some facts (or at least \surd). Terms of sort DQy are constructed with

$$\begin{aligned} \surd & : \rightarrow \text{DQy}^s, \quad _ \triangleright _ : \text{DQy DQy} \rightarrow \text{DQy}, \quad _ \triangleright _ : \text{Qy Qy} \rightarrow \text{Qy}, \quad _ \triangleright _ : \text{DQy}^s \text{DQy} \rightarrow \text{DQy}^s, \\ _ \triangleright _ & : \text{DQy DQy}^s \rightarrow \text{DQy}^s, \quad \emptyset : \rightarrow \text{Qy}, \quad _ \Rightarrow _ : \text{Cnd DQy} \rightarrow \text{DQy}, \quad _ \Rightarrow _ : \text{Cnd Qy} \rightarrow \text{Qy}, \\ \nabla _ _ & : \text{Pat}^t \text{DQy} \rightarrow \text{DQy}, \quad \nabla _ _ : \text{Pat}^p \text{Qy} \rightarrow \text{Qy}, \quad \nabla _ _ : \text{Pat}^{sf} \text{DQy}^s \rightarrow \text{DQy}^s. \end{aligned}$$

Thus, $\emptyset, f, Q \triangleright Q', \phi \Rightarrow Q$, and $\nabla P. Q$ are queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ if f is a fact, P is a terminating and preserving pattern, Q and Q' are queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$, and ϕ is a condition. Similarly, $\emptyset, \surd, f, D \triangleright D', \phi \Rightarrow D$, and $\nabla P. D$ are DML queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ if f is a fact, P is a terminating pattern, D and D' are DML queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$, and ϕ is a condition. In addition, $\nabla P. D$ is a DML query if P is a more general semi-terminating pattern, but D is success-assured, i.e., either \surd , a fact, or of the form $D_1 \triangleright D_2$ where at least one of D_1, D_2 is success assured. We force D to be success assured when P is semi-terminating, but not terminating because quantification over semi-terminating pattern may not terminate if the quantified expression can fail. Informal semantics of (DML) queries is given by:

1. Let $f : \text{Fact}$. A query f returns f . A DML query f adds f to the current database.
2. \surd is used to mark a branch of a DML query as successful even if it does not add any facts.
3. \emptyset is a query returning the empty multiset of facts or a DML query which does nothing.
4. A query $Q \triangleright Q'$ returns the multiset union of results of Q and Q' . A DML query $D \triangleright D'$ adds to and removes from the database a multiset union of what D and D' add and remove, respectively. Since facts are removed immediately, $_ \triangleright _$ is not commutative for DML queries.
5. A query $\phi \Rightarrow Q$ returns what Q returns if ϕ is satisfied. It returns \emptyset otherwise. A DML query $\phi \Rightarrow D$ does what D does if ϕ is satisfied. It does nothing otherwise.
6. Quantifier $\nabla P. Q$ denotes iteration over facts in the database matching P . At each iteration step $\sigma(Q)$ is executed, where σ is the matching substitution. If P contains fresh facts, execution of a DML query $\nabla P. Q$ may introduce fresh values.

Let $cl?(Q)$ if and only if the query ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ (or $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$) is closed. Similarly as in the case of conditions, $cl?(Q) := cl?(Q, \emptyset)$, where, for any set of variables V ,

$$\begin{aligned} cl?(\emptyset, V) &= cl?(\surd, V) = \mathbf{t}, \quad cl?(\phi \Rightarrow Q, V) = cl?(\phi, V) \wedge cl?(Q, V), \\ cl?(f, V) &= \text{Var}(f) \subseteq V \text{ if } f : \text{Fact} \\ cl?(Q_1 \triangleright Q_2, V) &= cl?(Q_1, V) \wedge cl?(Q_2, V), \quad cl?(\nabla P. Q, V) = cl?(Q, V \cup \text{Var}(P)). \end{aligned}$$

Definition 3.5. A (DML) subquery of a (DML) query Q is a subterm of Q of sort $(\text{D})\text{Qy}$.

Example 3.6. The following query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ is closed:

$$\nabla [f(x) \circ g]_? \circ [h(x)]! . (\{x > 5\} \Rightarrow f(x+1)).$$

It returns a multiset of facts consisting of facts of the form $f(x + 1)$ where $f(x)$ and $h(x)$ belong to the multiset we query and $x > 5$. When evaluating the query each source fact $f(x)$ (such that $h(x)$ is in the multiset and $x > 5$) and some fact g are matched only once during the evaluation of this query (on the other hand, facts of the form $h(x)$ can be matched multiple times). Thus, we return at most n facts $f(x + 1)$, where n is the number of g facts in the database. For example, for multiset $f(7) \circ f(8) \circ f(7) \circ f(4) \circ h(8) \circ h(7) \circ g \circ g$ the query returns either $f(7) \circ f(8)$ or $f(7) \circ f(7)$ depending on whether we match $f(7) \circ g$ and $f(8) \circ g$ or we match $f(7) \circ g$ twice (and then, in both cases, we run out of g -facts).

Example 3.7. The following (closed) “DML query” is not syntactically correct (it cannot be assigned sort DQy):

$$\nabla[f(x)]_0 . (\{x > 5\} \Rightarrow g(x))$$

This is because the pattern $[f(x)]_0$ is only semiterminating, but not terminating. In this case the subquery in the scope of $\nabla[f(x)]_!$ should be success assured, but $\{x > 5\} \Rightarrow g(x)$ clearly isn’t (if $x \leq 5$ then it fails). If we would execute this query against a multiset $f(1) \circ f(6)$ it would loop forever matching $f(1)$, removing it, failing when executing $\{1 > 5\} \Rightarrow g(1)$, returning $f(1)$ to the multiset, matching it again, and so on.

As the following example demonstrates, some incorrect DML queries of the form $\nabla P . \psi$, where P is semiterminating but not terminating and ϕ is not success assured, provably always terminate. Nevertheless, we prefer to reject them regardless, since usually making them syntactically correct is not difficult:

Example 3.8. Assume that x is a variable of sort $\mathbb{N}at$ (natural numbers). Execution of the following syntactically incorrect DML query always terminates (it replaces each fact of the form $f(x)$ where $x > 5$ with fact $g(x)$, and returns to the multiset all other $f(x)$ ’s unchanged):

$$\nabla[f(x)]_0 . ((\{x > 5\} \Rightarrow g(x)) \triangleright (\{x \leq 5\} \Rightarrow f(x))).$$

Termination follows from the fact that $(\{x > 5\} \Rightarrow g(x)) \triangleright (\{x \leq 5\} \Rightarrow f(x))$ always succeeds regardless of which natural number is bound to x : depending on whether $x > 5$ or $x \leq 5$ (and one of these must be true) either left or right argument of \triangleright succeeds and hence the whole subquery succeeds.

Unfortunately, neither $\{x > 5\} \Rightarrow g(x)$ nor $\{x \leq 5\} \Rightarrow f(x)$ is syntactically success assured and hence $(\{x > 5\} \Rightarrow g(x)) \triangleright (\{x \leq 5\} \Rightarrow f(x))$ is also not success assured. However, it is very easy to modify the above query so that it describes the same modification to the database, but is now syntactically correct and can be assigned sort DQy:

$$\nabla[f(x)]_0 . ((\{x > 5\} \Rightarrow g(x)) \triangleright (\{x \leq 5\} \Rightarrow f(x)) \triangleright \surd).$$

4. Rewriting semantics of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$

Semantics of a sentence ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ is given by the rewriting system $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{cnd}}(\phi)$. Terms rewritten by $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{cnd}}(\phi)$ are of the form $\{F, S\}^c$, where $\{-, -\}^c : \text{FSet Stk}^c \rightarrow \text{State}^c$, F is the database of facts on

which ϕ is checked, and S is a stack of sort Stk^c implementing structural recursion. Normal forms, constructed with $\varepsilon : \text{Bool} \rightarrow \text{State}^c$, encapsulate the result of ϕ 's evaluation. Sort State^c of terms holding the full state of evaluation must be distinct from all the other sorts and it must not have any super- or sub-sort relation to other sorts. This guarantees that there are no constructors accepting terms of sort State^c as arguments. Consequently, if all rewrite rules have terms of this sort on the left-hand side, then no subterms can be rewritten, i.e., the defined rewriting system is top-level. Terms of sort Stk^c are built from frames of sort Frm^c , where $\text{Frm}^c < \text{Stk}^c$, using an associative binary operator $_ : \text{Stk}^c \text{Stk}^c \rightarrow \text{Stk}^c$ with identity \emptyset . Most constructors of frames are indexed by subconditions ψ of ϕ , and lists of distinct variable names $\vec{v} := v_1, \dots, v_n$ of respective sorts s_1, \dots, s_n (\vec{v} can be of any length, even be empty, as long as $\{\vec{v}\} \subseteq \text{Var}(\phi)$ and it contains free variables of ψ):

$$\begin{aligned} \tau : \text{Bool} \rightarrow \text{Frm}^c, \quad \neg : \rightarrow \text{Frm}^c, \quad [_ , \dots , _]_{\psi}^{\vec{v}|^c}, [_ , \dots , _]_{\psi}^{\vec{v}|^c, \downarrow} : s_1 \dots s_n \rightarrow \text{Frm}^c, \\ [_]_{\exists P, \psi}^{\vec{v}|^c} : \text{FSet } s_1 \dots s_n \rightarrow \text{Frm}^c \end{aligned}$$

As \vec{v} can be empty, the above signature templates include $[_]_{\psi}^{|^c}, [_]_{\psi}^{|^c, \downarrow} : \rightarrow \text{Frm}^c$ and $[_]_{\psi}^{|^c} : \text{FSet} \rightarrow \text{Frm}^c$. $\tau(B)$ encapsulates the result of evaluation of a subcondition. Frame \neg negates the result of the next frame on the stack. Frames of the form $[\vec{a}]_{\psi}^{\vec{v}|^c}$, $[\vec{a}]_{\psi}^{\vec{v}|^c, \downarrow}$, or $[F' | \vec{a}]_{\psi}^{\vec{v}|^c}$ are called (ψ, σ) -frames, where $\sigma := \{\vec{a}/\vec{v}\}$ is the current substitution. They are related to evaluation of $\sigma(\psi)$. Marked frames $[\vec{a}]_{\psi}^{\vec{v}|^c, \downarrow}$ occur in evaluation of disjunctions $_ \vee _$. Iterator frames $[F' | \vec{a}]_{\exists P, \psi}^{\vec{v}|^c}$ represent iterative evaluation of quantifiers. Multiset F' , called iterator state, contains facts available for matching with $P_1 \circ P_2$. Given a database F , to evaluate sentence ϕ we rewrite a state term $I_{\phi}(F) := \{F, [_]_{\phi}^{|^c}\}$ until a normal form $\varepsilon(B)$ is reached. If B then ϕ is satisfied in F .

Now we present rule schemata for $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{cnd}}(\phi)$ instantiated for a given formula ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}$. It is important to distinguish between object variables substituted when applying actual rules, and metavariables used to define rule templates. Below, $F, F' : \text{FSet}$, $S : \text{Stk}^c$, and $B : \text{Bool}$ are object variables. We also denote by $\vec{v} := v_1, \dots, v_n$ and $\vec{w} = w_1, \dots, w_m$ sequences of object variables in $\text{Var}(\phi)$. Metavariables, ψ, ψ_1 , and ψ_2 stand for arbitrary subconditions of ϕ , metavariable \mathcal{B} stands for Boolean subterms of ϕ . Metavariable P stands for arbitrary patterns in ϕ , and P_1 and P_2 denote multisets of facts in the instance of pattern P marked by respective modalities (see Remark 2.3).

Constant \perp evaluates to \mathbf{f} , and $\{\mathcal{B}\}$ evaluates to $\sigma(\mathcal{B})$, where σ is the current substitution of variables \vec{v} ($\sigma(\mathcal{B})$ is ground since $\{\vec{v}\}$ contains all free variables (i.e., all variables) of \mathcal{B} , hence, by assumption, it simplifies to either \mathbf{f} or \mathbf{t}):

$$\lambda_{\perp} : \{F, S[\vec{v}]_{\perp}^{\vec{v}|^c}\}^c \Rightarrow \{F, \text{St}(\mathbf{f})\}^c, \quad \lambda_{\{_ \}} : \{F, S[\vec{v}]_{\{\mathcal{B}\}}^{\vec{v}|^c}\}^c \Rightarrow \{F, \text{St}(\mathcal{B})\}^c. \quad (3)$$

To evaluate $\sigma(\neg\psi)$, we “unfold” by replacing the $(\neg\psi, \sigma)$ -frame with \neg and (ψ, σ) -frame. When $\sigma(\psi)$ is evaluated we “fold” by negating the result:

$$\lambda_{\neg}^{\text{unf}} : \{F, S[\vec{v}]_{\neg\psi}^{\vec{v}|^c}\}^c \Rightarrow \{F, S\neg[\vec{v}]_{\psi}^{\vec{v}|^c}\}^c, \quad \lambda_{\neg}^{\text{fld}} : \{F, S\neg\tau(\mathcal{B})\}^c \Rightarrow \{F, \text{St}(\neg\mathcal{B})\}^c. \quad (4)$$

Remark 4.1. In our notation the same symbols often play a dual role — as subterms, and as part of function symbols (in sub- and super-scripts). Consider the following instantiation of schema $\lambda_{\neg}^{\text{unf}}$:

$$\lambda_{\neg}^{\text{unf}} : \{F, S[x, y]_{\neg\{x=y\}}^{x,y|c}\}^c \Rightarrow \{F, S\neg[x, y]_{\{x=y\}}^{x,y|c}\}^c$$

Variables in sub- and super-scripts are never substituted: with the above rule we have a one step rewrite of ground terms $\{\emptyset, [0, 1]_{\neg\{x=y\}}^{x,y|c}\}^c \xrightarrow{\lambda_{\neg}^{\text{unf}}} \{\emptyset, \neg[0, 1]_{\{x=y\}}^{x,y|c}\}^c$. In schema $\lambda_{\{-\}}$ metavariable \mathcal{B} occurs both as part of a name and as a subterm. An instantiation $\lambda_{\{-\}} : \{F, S[x, y]_{\{x=y\}}^{x,y|c}\}^c \Rightarrow \{F, S\tau(x = y)\}^c$ yields (with $S = \neg$, $x = 0$, and $y = 1$) a one step rewrite $\{\emptyset, \neg[0, 1]_{\{x=y\}}^{x,y|c}\}^c \xrightarrow{\lambda_{\{-\}}} \{\emptyset, \neg\tau(0 = 1)\}^c = \{\emptyset, \neg\tau(\mathbf{f})\}^c$.

To evaluate disjunction $\psi_1 \vee \psi_2$ we create two frames corresponding to the disjuncts. If ψ_2 evaluates to \mathbf{t} , the frame marked by \downarrow is dropped (disjunctions are short circuited). If ψ_2 evaluates to \mathbf{f} , the frame corresponding to ψ_1 drops \downarrow and is evaluated normally.

$$\begin{aligned} \lambda_{\vee}^{\text{unf}} : \{F, S[\vec{v}]_{\psi_1 \vee \psi_2}^{\vec{v}|c}\}^c &\Rightarrow \{F, S[\vec{v}]_{\psi_1}^{\vec{v}|c}, \downarrow [\vec{v}]_{\psi_2}^{\vec{v}|c}\}^c, \\ \lambda_{\vee; \mathbf{t}}^{\text{fld}} : \{F, S[\vec{v}]_{\psi}^{\vec{v}|c}, \downarrow \tau(\mathbf{t})\}^c &\Rightarrow \{F, S\tau(\mathbf{t})\}^c, \quad \lambda_{\vee; \mathbf{f}}^{\text{fld}} : \{F, S[\vec{v}]_{\psi}^{\vec{v}|c}, \downarrow \tau(\mathbf{f})\}^c \Rightarrow \{F, S[\vec{v}]_{\psi}^{\vec{v}|c}\}^c, \end{aligned} \quad (5)$$

Quantifier evaluation is initialized with the whole database available for matching:

$$\lambda_{\exists}^{\text{init}} : \{F, S[\vec{v}]_{\exists P, \psi}^{\vec{v}|c}\}^c \Rightarrow \{F, S[F | \vec{v}]_{\exists P, \psi}^{\vec{v}|c}\}^c. \quad (6)$$

Let \vec{w} be a sequence of all the distinct variables in $\text{Var}(P) \setminus \{\vec{v}\}$. Let σ be the current substitution. Rule $\lambda_{\exists}^{\text{unf}}$ pushes onto the stack a (σ', ψ) -frame, where $\sigma' = \sigma \cup \{\vec{b}/\vec{w}\}$ is defined by matching $F' \circ \sigma(P_! \circ P_?)$ with iterator state, and it removes $\sigma'(P_?)$ from the iterator state. We keep applying $\lambda_{\exists}^{\text{unf}}$ until $\sigma'(\psi)$ evaluates to \mathbf{t} or we cannot match $\sigma(P_! \circ P_?)$ with iterator state:

$$\begin{aligned} \lambda_{\exists}^{\text{unf}} : \{F, S[F' \circ P_! \circ P_? | \vec{v}]_{\exists P, \psi}^{\vec{v}|c}\}^c &\Rightarrow \{F, S[F' \circ P_! | \vec{v}]_{\exists P, \psi}^{\vec{v}|c}, [\vec{v}, \vec{w}]_{\psi}^{\vec{v}, \vec{w}|c}\}^c, \quad (7) \\ \lambda_{\exists; \mathbf{t}}^{\text{fld}} : \{F, S[F' | \vec{v}]_{\exists P, \psi}^{\vec{v}|c}, \tau(\mathbf{f})\}^c &\Rightarrow \{F, S[F' | \vec{v}]_{\exists P, \psi}^{\vec{v}|c}\}^c, \quad \lambda_{\exists; \mathbf{f}}^{\text{fld}} : \{F, S[F' | \vec{v}]_{\exists P, \psi}^{\vec{v}|c}, \tau(\mathbf{t})\}^c \Rightarrow \{F, S\tau(\mathbf{t})\}^c. \end{aligned}$$

Let Yes? be a sort and let $\text{yes} : \rightarrow \text{Yes?}$. For all $\vec{v} \subseteq \text{Var}(\phi)$ and patterns P occurring in ϕ we define a function $\mu_{P, \vec{v}} : \text{Pat}^{\text{tp}} s_1 \dots s_n \rightarrow \text{Yes?}$ with the single equation

$$\mu_{P, \vec{v}}(F' \circ P_! \circ P_?, \vec{v}) = \text{yes}. \quad (8)$$

Thus, $\mu_{P, \vec{v}}(F, \vec{a}) = \text{yes}$ if and only if F matches with $F' \circ \{\vec{a}/\vec{v}\}(P_? \circ P_0)$. Since facts matched by $\sigma(P_?)$ are removed from the iteration state, $\lambda_{\exists}^{\text{unf}}$ cannot be applied infinitely many times. The following rule schema makes $\sigma(\exists P, \psi)$ evaluate to \mathbf{f} when $\lambda_{\exists}^{\text{unf}}$ can no longer be applied:

$$\lambda_{\exists}^{\text{end}} : \{F, S[F' | \vec{v}]_{\exists P, \psi}^{\vec{v}|c}\}^c \Rightarrow \{F, S\tau(\mathbf{f})\}^c \quad \text{if } (\mu_{P, \vec{v}}(F', \vec{v}) = \text{yes}) = \mathbf{f}. \quad (9)$$

Finally, the rule $\lambda_{\text{sat}} : \{F, \tau(B)\}^c \Rightarrow \mathfrak{s}(B)$ finishes evaluation of ϕ .

Theorem 4.2. $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{end}}(\phi)$ is a terminating rewriting system.

Proof:

It suffices to define a partial well-order $_ <_c _$ on terms of sort State^c which makes rewriting strictly monotonic, i.e., such that $t_1 \rightarrow t_2$ implies $t_2 <_c t_1$ for all $t_1, t_2 : \text{State}^c$. The order is defined by

$$\mathfrak{s}(B) <_c \{F, S\}^c, \quad \{F, S\}^c <_c \{F, S'\}^c \text{ iff } S <_s S',$$

for all F, S, S', B . Here $_ <_s _$ is the lexicographic order on stacks derived from partial order $_ <_f _$ on frames, i.e., for all frames $D_1, \dots, D_n, E_1, \dots, E_m, D_1 \dots D_n <_s E_1 \dots E_m$ iff either (1) for some k , $D_k <_f E_k$ and $D_i = E_i$ for $i \in \{1, \dots, k-1\}$, or (2) $n < m$ and $D_i = E_i$ for $i \in \{1, \dots, n\}$. Frame ordering is defined by $\mathfrak{r}(B) <_f \neg <_f [F \mid \vec{a}]_{\psi}^{\vec{v}|c} <_f [F' \mid \vec{a}]_{\psi}^{\vec{v}|c} <_f [\vec{b}]_{\psi}^{\vec{w}|c} <_f [\vec{b}]_{\psi}^{\vec{w}|c, \downarrow} <_f [F'' \mid \vec{c}]_{\psi'}^{\vec{x}|c}$, for all $F, F', F'', \psi, \psi', B, \vec{v}, \vec{w}, \vec{x}, \vec{a}, \vec{b}, \vec{c}$ such that $F \not\subseteq F'$ and ψ is a proper subcondition of ψ' . Partial order $_ <_f _$ is Noetherian because multisets of facts F, F' and conditions ψ, ψ' are finite terms. Hence, if the stacks are of bounded size, also $_ <_c _$ is Noetherian. The size of stacks is bounded because each stack size increasing rule is of the form $\{F, S[\dots]_{\psi_1}^{\dots}\}^c \rightarrow \{F, SA[\dots]_{\psi_2}^{\dots}\}^c$, where A is a frame and ψ_2 is a proper subterm of ψ_1 . Since rules in $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{end}}(\phi)$ are topmost, the rewriting is strictly monotonic because $t_2 <_c t_1$ for each rule schema $t_1 \Rightarrow t_2$ if C in $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{end}}(\phi)$. \square

The following useful observation can be trivially verified by examining the rule schemas:

Lemma 4.3. Let ψ be a subcondition of ϕ . For any finite multiset of facts F , stack S , Boolean B , variables $\vec{v} = v_1, \dots, v_n$ and values $\vec{a} = a_1, \dots, a_n$, $\{F, S[\vec{a}]_{\psi}^{\vec{v}|c}\}^c \rightarrow^* \{F, S\mathfrak{r}(B)\}^c$ in $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{end}}(\phi)$ iff $\{F, \llbracket \sigma(\psi) \rrbracket^c\}^c \rightarrow^* \mathfrak{s}(B)$ in $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{end}}(\sigma(\psi))$, where substitution $\sigma := \{\vec{a}/\vec{v}\}$.

The following example shows verification of a condition in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{end}}$ for a given multiset of facts using rewriting semantics. It also shows non-confluence of the resulting rewrite system.

Example 4.4. Suppose $p, q : \text{Nat} \rightarrow \text{Fact}$. Let $\psi := \exists [q(z)]?. \{z = x\}$, $K := p(1) \circ q(0)$, $L := p(0) \circ q(0)$, and $M := p(0) \circ p(0) \circ p(1)$. To check if condition

$$\phi := \exists [p(x) \circ p(y)]? \circ [q(y)]! \cdot (\neg\psi \vee \{x = y\}).$$

is satisfied in a multiset $H := p(0) \circ p(0) \circ K = L \circ p(0) \circ p(1) = M \circ q(0)$ we normalize

$$\begin{aligned} I_{\phi}(H) &\xrightarrow{\lambda_{\exists}^{\text{init}}} \{H, [H]_{\phi}^c\}^c \xrightarrow{\lambda_{\exists}^{\text{unf}}} \{H, [K]_{\phi}^c [0, 0]_{\neg\psi \vee \{x=y\}}^{x,y|c}\}^c \xrightarrow{\lambda_{\forall}^{\text{unf}}} \{H, [K]_{\phi}^c [0, 0]_{\neg\psi}^{x,y|c, \downarrow} [0, 0]_{\{x=y\}}^{x,y|c}\}^c \\ &\xrightarrow{\lambda_{\{\downarrow\}}^c} \{H, [K]_{\phi}^c [0, 0]_{\neg\psi}^{x,y|c, \downarrow} \mathfrak{r}(0=0)\}^c \xrightarrow{\lambda_{\text{it}}^{\text{nd}}} \{H, [K]_{\phi}^c \mathfrak{r}(\mathbf{t})\}^c \xrightarrow{\lambda_{\text{it}}^{\text{nd}}} \{H, \mathfrak{r}(\mathbf{t})\}^c \xrightarrow{\lambda_{\text{sat}}} \mathfrak{s}(\mathbf{f}). \end{aligned}$$

Thus, ϕ is satisfied in H . However, we have also a normalizing sequence ending with $\mathfrak{s}(\mathbf{f})$:

$$\begin{aligned} I_{\phi}(H) &\xrightarrow{\lambda_{\exists}^{\text{init}}} \{H, [H]_{\phi}^c\}^c \xrightarrow{\lambda_{\exists}^{\text{unf}}} \{H, [L]_{\phi}^c [0, 1]_{\neg\psi \vee \{x=y\}}^{x,y|c}\}^c \rightarrow^* \{H, [L]_{\phi}^c [0, 1]_{\neg\psi}^{x,y|c}\}^c \\ &\rightarrow^* \{H, [L]_{\phi}^c \neg [H \mid 0, 1]_{\psi}^{x,y|c}\}^c \xrightarrow{\lambda_{\exists}^{\text{unf}}} \{H, [L]_{\phi}^c \neg [M \mid 0, 1]_{\psi}^{x,y|c} [0, 1, 0]_{\{z=x\}}^{x,y,z|c}\}^c \\ &\rightarrow^* \{H, [L]_{\phi}^c \neg \mathfrak{r}(\mathbf{t})\}^c \rightarrow^! \mathfrak{s}(\mathbf{f}). \end{aligned}$$

Thus, evaluation of conditions does not necessarily lead to a unique result (the rewriting system is not confluent). This requires making the definition of logical equivalence bisimulation-like:

Definition 4.5. Let ϕ_1 and ϕ_2 be conditions in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$. We say that ϕ_1 is logically equivalent to ϕ_2 , writing $\phi_1 \equiv \phi_2$, if and only if for all ground multisets of facts F , ground substitutions σ such that $\sigma(\phi_1)$ and $\sigma(\phi_2)$ are closed, and a Boolean $B \in \{\mathbf{t}, \mathbf{f}\}$ we have

$$I_{\sigma(\phi_1)}(F) \rightarrow^! \mathfrak{s}(B) \quad \text{if and only if} \quad I_{\sigma(\phi_2)}(F) \rightarrow^! \mathfrak{s}(B).$$

The following result is an immediate consequence of Lemma 4.3:

Lemma 4.6. Logical equivalence on conditions in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ is an equivalence and a congruence, i.e., if κ is a position in a condition ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ such that $\phi|_{\kappa}$ is a condition, and $\psi \equiv \phi|_{\kappa}$, then $\phi \equiv \phi[\psi]_{\kappa}$.

A renaming is an injective substitution σ mapping variables to variables.

Lemma 4.7. For any closed condition ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$, and any renaming σ , $\phi \equiv \sigma(\phi)$.

The following result clarifies elements of rewriting semantics of sentences in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$:

Lemma 4.8. For each ground multiset of facts F , and all sentences ϕ, ϕ_1, ϕ_2 and $\exists P. \psi$:

1. $I_{\phi}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ or $I_{\phi}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$, and these are the only possible normal forms of $I_{\phi}(F)$.
2. $I_{\perp}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$ and never $I_{\perp}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$.
3. $I_{\phi}(F) \rightarrow^! \mathfrak{s}(B)$ iff $I_{\neg\phi}(F) \rightarrow^! \mathfrak{s}(\neg B)$ for all $B : \text{Bool}$.
4. $I_{\phi_1 \vee \phi_2}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ iff $I_{\phi_1}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ or $I_{\phi_2}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$.
5. $I_{\phi_1 \vee \phi_2}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$ iff $I_{\phi_1}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$ and $I_{\phi_2}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$.
6. $I_{\exists P. \psi}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ iff there exists a substitution σ , and a multiset F' such that $F' \circ \sigma(P_{?} \circ P_{?}) = F$ and $I_{\sigma(\psi)}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$.
7. $I_{\exists P. \psi}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$ iff there exist two sequences of ground multisets of facts F_0, F_1, \dots, F_n and G_0, G_1, \dots, G_{n-1} , and a sequence of substitutions $\sigma_0, \sigma_1, \dots, \sigma_{n-1}$ such that
 - (a) $F_0 = F, F_{i+1} = G_i \circ \sigma_i(P_i)$, and $F_i = G_i \circ \sigma_i(P_i \circ P_{?})$, for all $i \in \{0, \dots, n-1\}$,
 - (b) $I_{\sigma_i(\psi)}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$, for all $i \in \{0, \dots, n-1\}$,
 - (c) there exists no substitution σ_n and multiset of facts G_n such that $F_n = G_n \circ \sigma_n(P_i \circ P_{?})$.
8. $I_{\phi \vee \neg\phi}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$. If both $I_{\phi}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ and $I_{\phi}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$ then also $I_{\phi \vee \neg\phi}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$.

Proof:

The first point is verified by structural recursion using Lemma 4.3 and rules in Equations (3)–(9). Points 2–7 are verified using rules in Equations (3)–(9). Point 8 is verified using points 3–5. \square

Lemma 4.9. The following logical equivalences hold between conditions in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$:

$$\begin{aligned} \phi \vee \perp &\equiv \phi, & \phi_1 \vee \phi_2 &\equiv \phi_2 \vee \phi_1, & \phi_1 \vee (\phi_2 \vee \phi_3) &\equiv (\phi_1 \vee \phi_2) \vee \phi_3, \\ \neg\neg\phi &\equiv \phi, & \exists P. (\phi_1 \vee \phi_2) &\equiv (\exists P. \phi_1) \vee (\exists P. \phi_2). \end{aligned}$$

Proof:

The above equivalences can be proven using points 1-7 in Lemma 4.8. Only the last equivalence's proof is non-trivial. Let F be a ground multiset of facts and let σ be a ground substitution such that $\sigma(\exists P. (\phi_1 \vee \phi_2))$ (or, equivalently, $\sigma((\exists P. \phi_1) \vee (\exists P. \phi_2))$) is closed. Denote $Q := \sigma(P)$, $\psi_i := \sigma(\phi_i)$, for $i \in \{1, 2\}$. Using Lemma 4.8, p. 6, we see that $I_{\exists Q. (\psi_1 \vee \psi_2)}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ iff there exists a substitution σ' , and a multiset of facts F' such that $F' \circ \sigma'(Q? \circ Q!) = F$ and $I_{\sigma'(\psi_1) \vee \sigma'(\psi_2)}(F') \rightarrow^! \mathfrak{s}(\mathbf{t})$. The latter holds iff there exists $i \in \{1, 2\}$ such that $I_{\sigma'(\psi_i)}(F') \rightarrow^! \mathfrak{s}(\mathbf{t})$, by Lemma 4.8, p. 4. It follows, again using Lemma 4.8, point 6, that $I_{\exists Q. (\psi_1 \vee \psi_2)}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ iff $I_{\exists Q. \psi_i}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$ for some $i \in \{1, 2\}$, i.e., iff (by Lemma 4.8, p. 4) $I_{(\exists Q. \psi_1) \vee (\exists Q. \psi_2)}(F) \rightarrow^! \mathfrak{s}(\mathbf{t})$. The part of the proof with falsity is more complex. Using Lemma 4.8, p. 7, we see that $I_{\exists Q. (\psi_1 \vee \psi_2)}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$ iff there exist two sequences of ground multisets of facts F_0, \dots, F_n and G_0, \dots, G_{n-1} , and a sequence of substitutions $\sigma_0, \dots, \sigma_{n-1}$ such that (a) $F_0 = F$, $F_{i+1} = G_i \circ \sigma_i(Q!)$ and $F_i = G_i \circ \sigma_i(Q! \circ Q?)$, for all $i \in \{0, \dots, n-1\}$, (b) $I_{\sigma_i(\psi_1) \vee \sigma_i(\psi_2)}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$, for all $i \in \{0, \dots, n-1\}$, (c) there exists no substitution σ_n and multiset of facts G_n such that $F_n = G_n \circ \sigma_n(Q! \circ Q?)$. Then by Lemma 4.8, p. 5, (b) iff, for $j \in \{1, 2\}$, (b_j) $I_{\sigma_i(\psi_j)}(F_i) \rightarrow^! \mathfrak{s}(\mathbf{f})$, for all $i \in \{0, \dots, n-1\}$. Then (by Lemma 4.8, p. 7) (a), (b₁), (b₂) and (c) iff $I_{\exists Q. \psi_j}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$ for $j \in \{1, 2\}$ iff, by Lemma 4.8, p. 5, $I_{(\exists Q. \psi_1) \vee (\exists Q. \psi_2)}(F) \rightarrow^! \mathfrak{s}(\mathbf{f})$. \square

Non-confluence of $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{cnd}}(\phi)$ in Example 4.4 depended on patterns in ϕ with more than one fact. It turns out that $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{cnd}}(\phi)$ may be non-confluent even if ϕ contains only single-fact patterns:

Example 4.10. Let $r : \text{Nat Nat} \rightarrow \text{Fact}$ be commutative. Consider condition $\phi := \exists[r(x, y)]?. \{x = 1\}$ evaluated in a database $F = r(1, 2)$. Since r is commutative, there are two distinct substitutions $\{1/x, 2/y\}$ and $\{2/x, 1/y\}$ which match $r(x, y)$ with $r(1, 2)$. Consequently, there are two distinct paths of evaluating ϕ : $I_\phi(F) \rightarrow^* \{F, [F]_\phi^c\} \xrightarrow{\lambda_{\exists}^{\text{unf}}} \begin{cases} \{F, [\emptyset]_\phi^c[1, 2]_{\{x=1\}}^{x, y|c}\}^c \rightarrow^! \mathfrak{s}(\mathbf{t}) \\ \{F, [\emptyset]_\phi^c[2, 1]_{\{x=1\}}^{x, y|c}\}^c \rightarrow^! \mathfrak{s}(\mathbf{f}) \end{cases}$.

Definition 4.11. A fully reduced term $t : \text{Fact}$ is said to have a *unique matching* property iff for any ground, fully reduced term $t' : \text{Fact}$ there exists at most one substitution σ such that $\sigma(t) =_A t'$.

Definition 4.12. A condition ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ is called *deterministic* if and only if all quantification patterns in ϕ contain only single facts with unique matching property.

The following theorem states that while evaluation of a deterministic condition is not itself deterministic, but its results are.

Theorem 4.13. Let ϕ be a deterministic condition in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$. Then $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{cnd}}(\phi)$ is confluent. In particular, given a ground multiset of facts F , there is a unique $B \in \{\mathbf{t}, \mathbf{f}\}$ such that $I_\phi(F) \rightarrow^! \mathfrak{s}(B)$.

Proof:

We argue by induction on the complexity of formulas indexing frames on the top of a stack. First observe that only terms of the form $t := \{F, S[F' \mid \vec{a}]_{\exists P, \psi}^c\}^c$ can be rewritten in a single step into two distinct terms. As semiconfluence implies confluence it suffices to prove that if $t' \leftarrow t \rightarrow^+ t''$ then there exists s such that $t' \rightarrow^* s \leftarrow^* t''$. By Lemmas 4.3 and 4.8, p. 1, a rewrite sequence $t \rightarrow^+ t''$ must (1) contain $\{F, S\tau(B)\}^c$, or (2) can be extended to the sequence ending in $\{F, S\tau(B)\}^c$. If we prove that $t' \rightarrow^* \{F, S\tau(B)\}^c$ then we can set $s = t''$ (if (1)) or $s = \{F, S\tau(B)\}^c$ (if (2)). It remains to prove that $t' \rightarrow^* \{F, S\tau(B)\}^c$. Under the theorem's assumption, $P = [f]_?$, where f is a fact with a unique matching property (Definition 4.11). Thus, if for some fact g in F' there exists a substitution σ_g extending $\{\vec{a}/\vec{v}\}$ such that $g = \sigma_g(f)$ and $I_{\sigma_g(\psi)}(F) \rightarrow \tau(\mathbf{t})$ (which implies, by inductive assumption, that $I_{\sigma_g(\psi)}(F) \not\rightarrow^* \tau(\mathbf{f})$), it is unique, and cannot be missed during evaluation of the iterator frame. Either such g exists, and then $t \rightarrow^* \{F, S\tau(\mathbf{t})\}^c$ but $t \not\rightarrow^* \{F, S\tau(\mathbf{f})\}^c$ (hence necessarily both $B = \mathbf{t}$ and $t' \rightarrow^* \{F, S\tau(\mathbf{t})\}^c$), or it doesn't, and hence, both $B = \mathbf{f}$ and $t' \rightarrow^* \{F, S\tau(\mathbf{f})\}^c$ \square

Example 4.14. Suppose $p : \text{Nat} \rightarrow \text{Fact}$, $q : \rightarrow \text{Fact}$, and let $r : \text{Nat Nat} \rightarrow \text{Fact}$ be a commutative operator (i.e., $r(x, y) =_A r(y, x)$). Clearly, $p(x)$ and q have unique matching property (Definition 4.11), while $r(x, y)$ does not, since, e.g., if $\sigma_1 := \{0/x, 1/y\}$ and $\sigma_2 := \{1/x, 0/y\}$ then $\sigma_1(r(x, y)) =_A \sigma_2(r(x, y)) =_A r(0, 1)$. Let

$$\phi_1 := \exists[p(x)]_? \cdot \{x = 0\}, \quad \phi_2 := \exists[p(x) \circ q]_? \cdot \{x = 0\}, \quad \phi_3 := \exists[r(x, y)]_? \cdot \{x = 0\}.$$

Then ϕ_1 is deterministic (Definition 4.12), while ϕ_2 and ϕ_3 are not deterministic. Let $F := p(0) \circ p(1) \circ q \circ r(0, 1)$. We now consider evaluation of all the ϕ_i 's on F . First, the reader will easily verify that while evaluating the existential quantifier in ϕ_1 we can either first match $p(x)$ with $p(1)$ and then, upon failure, with $p(0)$, or first match with $p(0)$. Eventually, both paths yield satisfaction of ϕ_1 on F (although the first path is longer). On the other hand, evaluation of ϕ_2 and ϕ_3 demonstrate two ways in which non-determinism occurs in evaluation of conditions in $\mathcal{Q}_{\Sigma, D}^{\text{end}}$. First, consider two non-convergent paths of rewriting $I_{\phi_2}(F)$:

$$\begin{array}{ccc} \{F, [p(1) \circ r(0, 1)]_{\phi_2}^c [0]_{\{x=0\}}^c\}^c & \xrightarrow{\lambda_{\{-\}}} & \{F, [p(1) \circ r(0, 1)]_{\phi_2}^c \tau(\mathbf{t})\}^c \xrightarrow{*} \mathbf{s}(\mathbf{t}) \\ \lambda_{\exists}^{\text{unf}} \uparrow & & \\ \{F, [p(0) \circ p(1) \circ q \circ r(0, 1)]_{\phi_2}^c\}^c & \xleftarrow{\lambda_{\exists}^{\text{init}}} I_{\phi_2}(F) & \mathbf{s}(\mathbf{f}) \\ \lambda_{\exists}^{\text{unf}} \downarrow & & \uparrow \\ \{F, [p(0) \circ r(0, 1)]_{\phi_2}^c [1]_{\{x=0\}}^c\}^c & \xrightarrow{\lambda_{\{-\}}} & \{F, [p(0) \circ r(0, 1)]_{\phi_2}^c \tau(\mathbf{f})\}^c \xrightarrow{\lambda_{\exists}^{\text{end}}} \{F, \tau(\mathbf{f})\}^c \end{array}$$

Here the reason for non-determinism which ultimately leads to non-convergent paths of execution is that when evaluating the existential quantifier at each attempt we have to consume both $p(x)$ -fact and q -fact: since there is only one q fact, if we start from wrong $p(x)$ -fact (i.e., $p(1)$), we do not get the second chance.

Denote for brevity $K := p(0) \circ p(1) \circ q$. Recall that $r(x, y) =_A r(y, x)$. In particular, $r(0, 1) =_A r(1, 0)$. Consider now two non-convergent paths of rewriting $I_{\phi_3}(F)$:

$$\begin{array}{ccccc}
 \{F, [K] \upharpoonright_{\phi_3}^c [0, 1]_{\{x=0\}}^{x,y|c}\}^c & \xrightarrow{\lambda_{\{-\}}^{\uparrow}} & \{F, [K] \upharpoonright_{\phi_3}^c \mathbf{r}(\mathbf{t})\}^c & \xrightarrow{*} & \mathbf{s}(\mathbf{t}) \\
 \uparrow \lambda_{\exists}^{\text{unf}} & & & & \\
 \{F, [K \circ r(0, 1)] \upharpoonright_{\phi_3}^c\}^c & \xleftarrow{\lambda_{\exists}^{\text{init}}} & I_{\phi_3}(F) & & \mathbf{s}(\mathbf{f}) \\
 \downarrow \lambda_{\exists}^{\text{unf}} & & & & \uparrow * \\
 \{F, [K] \upharpoonright_{\phi_3}^c [1, 0]_{\{x=0\}}^{x,y|c}\}^c & \xrightarrow{\lambda_{\{-\}}^{\uparrow}} & \{F, [K] \upharpoonright_{\phi_3}^c \mathbf{r}(\mathbf{f})\}^c & \xrightarrow{\lambda_{\exists}^{\text{end}}} & \{F, \mathbf{r}(\mathbf{f})\}^c
 \end{array}$$

Thus, in this case the reason of non-determinism leading to non-convergent paths was the possibility of two distinct matchings of $r(0, 1)$ with $r(x, y)$ given by $\{0/x, 1/y\}$ and $\{1/x, 0/y\}$.

5. Rewriting semantics of $Q_{\Sigma, \mathcal{D}}^{\text{qry}}$

Let Q be a query in $Q_{\Sigma, \mathcal{D}}^{\text{qry}}$. We associate with Q the rewriting system $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{qry}}(Q)$. Terms rewritten with the rules of the rewriting system $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{qry}}(Q)$ are of the form $\{F, F', S\}^q$, where $\{-, _, _ \}^q : \text{FSet FSet Stk}^q \rightarrow \text{State}^q$, F is a database of facts against which we issue the query, F' is a partial answer (i.e., an answer built so far in the rewriting process), and S is a stack of sort Stk^q which simulates structural recursion. Normal forms encapsulating an answer to Q are constructed with $\mathbf{a} : \text{FSet} \rightarrow \text{State}^q$. Terms of sort Stk^q are constructed from local computation frames of sort Frm^q , where $\text{Frm}^q < \text{Stk}^q$, using an associative binary operator $_ _ : \text{Stk}^q \text{Stk}^q \rightarrow \text{Stk}^q$ with identity element \emptyset . Constructors of frames are indexed by sub-queries R of Q , and lists of distinct variable names $\vec{v} := v_1, \dots, v_n$ of respective sorts s_1, \dots, s_n (\vec{v} can be of any length as long as $\{\vec{v}\} \subseteq \text{Var}(Q)$ and it contains all free variables of R):

$$\begin{aligned}
 [-, \dots, -]_R^{\vec{v}|q} : s_1 \dots s_n \rightarrow \text{Frm}^q, \quad [-, \dots, -]_R^{\vec{v}|q} : s_1 \dots s_n \text{Stk}^c \rightarrow \text{Frm}^q, \\
 [-]_R^{\vec{v}|q} : \text{FSet } s_1 \dots s_n \rightarrow \text{Frm}^q \text{ if } R = \nabla P. R'.
 \end{aligned}$$

As \vec{v} can be empty, the above signature templates include $\square_R^c : \rightarrow \text{Frm}^q$, etc. As in the case of $Q_{\Sigma, \mathcal{D}}^{\text{end}}$, variables in super- and sub-scripts are part of function names and are never matched or substituted — Remark 4.1 applies here and in the next section. Frames of the form $[\vec{a}]_R^{\vec{v}|q}$, $[\vec{a}|S]_R^{\vec{v}|q}$, or $[F'|\vec{a}]_R^{\vec{v}|q}$ are called (R, σ) -frames, where $\sigma := \{\vec{a}/\vec{v}\}$ is the current substitution. They indicate evaluation of $\sigma(R)$. Conditional frames $[\vec{a}|S]_R^{\vec{v}|q}$ are used in evaluation of conditionals $\phi \Rightarrow R$, where $S : \text{Stk}^c$ represents evaluation of ϕ . Iterator frames $[F'|\vec{a}]_{\nabla P. R}^{\vec{v}|q}$, represent iterative evaluation of $\sigma(\nabla P. R)$. Multiset F' , called iterator state, contains facts available for matching with $P_1 \circ P_2$. Given a database F , to evaluate a closed query Q we rewrite a state term $I_Q(F) := \{F, \emptyset, \square_Q^c\}^q$ until a normal form $\mathbf{a}(F')$ is reached. Then we conclude that evaluation of Q on F yields F' as an answer.

Now we are ready to define the rules of $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{qry}}(Q)$. Literal facts are added to the partial answer multiset after applying the current substitution, and empty queries return nothing:

$$\lambda_{\text{fact}} : \{F, F', S[\vec{v}]_f^{\vec{v}|q}\}^q \Rightarrow \{F, F' \circ f, S\}^q, \quad \lambda_{\emptyset} : \{F, F', S[\vec{v}]_{\emptyset}^{\vec{v}|q}\}^q \Rightarrow \{F, F', S\}^q. \quad (10)$$

Evaluation of “union” $_ \triangleright _$ is implemented by replacing the frame corresponding to $R_1 \triangleright R_2$ with two frames corresponding to R_1 and R_2 , respectively:

$$\lambda_{\triangleright}^{\text{unf}} : \{F, F', S[\vec{v}]_{R_1 \triangleright R_2}^{\vec{v}|q}\}^q \Rightarrow \{F, F', S[\vec{v}]_{R_2}^{\vec{v}|q} [\vec{v}]_{R_1}^{\vec{v}|q}\}^q \quad (11)$$

To compute a conditional $\phi \Rightarrow R$ we first embed a stack representing computation of condition ϕ within the frame corresponding to the conditional. Once this condition is evaluated, we either evaluate R if the condition is satisfied, or drop the conditional if it is not:

$$\begin{aligned} \lambda_{\text{cond}}^{\text{unf}} &: \{F, F', S[\vec{v}]_{\phi \Rightarrow R}^{\vec{v}|q}\}^q \Rightarrow \{F, F', S[\vec{v} \mid [\vec{v}]_{\phi}^{\vec{v}|c}]_R^{\vec{v}|q}\}^q, \\ \lambda_{\text{cond}; \mathbf{f}}^{\text{unf}} &: \{F, F', S[\vec{v} \mid \mathbf{f}]_R^{\vec{v}|q}\}^q \Rightarrow \{F, F', S\}^q, \\ \lambda_{\text{cond}; \mathbf{t}}^{\text{unf}} &: \{F, F', S[\vec{v} \mid \mathbf{t}]_R^{\vec{v}|q}\}^q \Rightarrow \{F, F', S[\vec{v}]_R^{\vec{v}|q}\}^q. \end{aligned} \quad (12)$$

To compute ϕ we add, for every rule $\lambda : \{F, S'\}^c \Rightarrow \{F, S''\}^c$ if C in $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{end}}(\phi)$ the rule schema

$$\lambda^q : \{F, F', S[\vec{v} \mid S']_R^{\vec{v}|q}\}^q \Rightarrow \{F, F', S[\vec{v} \mid S'']_R^{\vec{v}|q}\}^q \text{ if } C \quad (13)$$

Evaluation of $\nabla _ _$ subquery is initialized with the whole database available for matching:

$$\lambda_{\nabla}^{\text{init}} : \{F, F', S[\vec{v}]_{\nabla P.R}^{\vec{v}|q}\}^q \Rightarrow \{F, F', S[F \mid \vec{v}]_{\nabla P.R}^{\vec{v}|q}\}^q \quad (14)$$

Let \vec{w} be a sequence of all the distinct variables in $\text{Var}(P) \setminus \{\vec{v}\}$. Let σ be the current substitution. Rule $\lambda_{\nabla}^{\text{unf}}$ pushes onto the stack a (σ', R) -frame, where $\sigma' = \sigma \cup \{\vec{b}/\vec{w}\}$ is defined by matching $F'' \circ \sigma(P_! \circ P_?)$ with iterator state, and it removes $\sigma'(P_?)$ from the iterator state:

$$\lambda_{\nabla}^{\text{unf}} : \{F, F', S[F'' \circ P_! \circ P_? \mid \vec{v}]_{\nabla P.R}^{\vec{v}|q}\}^c \Rightarrow \{F, F', S[F'' \circ P_! \mid \vec{v}]_{\nabla P.R}^{\vec{v}|q} [\vec{v}, \vec{w}]_R^{\vec{v}, \vec{w}|q}\}^q \quad (15)$$

We keep applying $\lambda_{\nabla}^{\text{unf}}$ until we cannot match $F'' \circ \sigma(P_! \circ P_?)$ with iterator state. Then we remove the iterator frame from the stack. To prevent premature application, rule schema $\lambda_{\nabla}^{\text{end}}$ is conditional, where the condition uses functions $\mu_{P, \vec{v}} : \text{Pat}^{\text{tp}} s_1 \dots s_n \rightarrow \text{Yes?}$ defined for each $\vec{v} \subseteq \text{Var}(Q)$ and pattern P occurring in Q with the single equation $\mu_{P, \vec{v}}(F \circ P_! \circ P_?, \vec{v}) = \text{yes}$ (cf. Equation (8)):

$$\lambda_{\nabla}^{\text{end}} : \{F, F', S[F'' \mid \vec{v}]_{\nabla P.R}^{\vec{v}|q}\}^q \Rightarrow \{F, F', S\}^q \quad \text{if } (\mu_{P, \vec{v}}(F'', \vec{v}) = \text{yes}) = \mathbf{f}. \quad (16)$$

Finally, the rule $\lambda_{\text{ans}} : \{F, F', \emptyset\}^c \Rightarrow \mathbf{a}(F')$ finishes evaluation of Q .

The following result can be proven similarly to Theorem 4.2:

Theorem 5.1. $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{qry}}(R)$ is a terminating rewriting system.

The following useful observation can be trivially verified by examining the rule schemas:

Lemma 5.2. Let R be a subcondition of Q . Then, for all multisets of facts F, F', F'' , stacks S , lists of variables $\vec{v} = v_1, \dots, v_n$ and values $\vec{a} = a_1, \dots, a_n$, $\{F, F', S[\vec{a}]^{\vec{v}|q}\}^q \rightarrow^* \{F, F' \circ F'', S\}^q$ in $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{qry}}(Q)$ if and only if $\{F, \emptyset, \llbracket \sigma(R) \rrbracket^q\}^q \rightarrow^! \mathbf{a}(F'')$ in $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{qry}}(\sigma(R))$, where substitution $\sigma := \{\vec{a}/\vec{v}\}$.

The following example shows that evaluation of queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ does not, in general, return a unique answer, and, in particular, that $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{qry}}(Q)$ for general queries Q is not confluent.

Example 5.3. Let $\sharp : \rightarrow \text{Fact}$ and $b : \text{Nat} \rightarrow \text{Fact}$. Consider the query $Q := \nabla[\sharp]_? \circ [b(x)]_! \cdot b(x)$ executed against database $F := \sharp \circ b(1) \circ b(2)$. Then $I_Q(F)$ can be normalized in two ways:

$$I_Q(F) \xrightarrow{\lambda_{\nabla}^{\text{ini}}} \{F, \emptyset, [F \mid]_Q^q\}^q \xrightarrow{\lambda_{\nabla}^{\text{unf}}} \begin{cases} \{F, \emptyset, [b(1) \circ b(2) \mid]_Q^q[1]_{b(x)}^q\}^q \rightarrow^! \mathbf{a}(b(1)) \\ \{F, \emptyset, [b(1) \circ b(2) \mid]_Q^q[2]_{b(x)}^q\}^q \rightarrow^! \mathbf{a}(b(2)) \end{cases} .$$

Queries like Q are useful as a simulation of a non-deterministic choice (say, by a human agent) of a subset of values stored in the database with a fixed maximal cardinality. E.g., $\nabla[b(x)]_? \cdot b(x)$ returns all “ b -facts” stored in the database. Query Q defined above, however, chooses (with repetitions) at most as many b -facts as there are tokens \sharp . Query $\nabla[\sharp \circ b(x)]_? \cdot b(x)$ avoids repetitions.

The following is the bisimulation-like definition of logical equivalence between queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$:

Definition 5.4. Let Q_1 and Q_2 be two conditions in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$. Recall that $I_Q(F) := \{F, \emptyset, \llbracket Q \rrbracket^q\}^q$. We say that Q_1 is logically equivalent to Q_2 , writing $Q_1 \equiv Q_2$, if and only if, for all ground multisets of facts F and F' , and ground substitutions σ such that $\sigma(Q_1)$ and $\sigma(Q_2)$ are closed, we have

$$I_{\sigma(Q_1)}(F) \rightarrow^! \mathbf{a}(F') \quad \text{iff} \quad I_{\sigma(Q_2)}(F) \rightarrow^! \mathbf{a}(F').$$

In other words, queries are equivalent if they can match each other’s answers. The following result is an immediate consequence of Lemma 5.2:

Lemma 5.5. Logical equivalence on queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ is an equivalence relation and a congruence, i.e., if κ is a position in a query Q in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ such that $Q|_{\kappa}$ is a query, and $R \equiv Q|_{\kappa}$, then $Q \equiv Q[R]_{\kappa}$.

We leave proof of the next observation to the reader:

Lemma 5.6. For any closed query Q in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$, and any renaming σ , $Q \equiv \sigma(Q)$.

The following clarification of semantics of queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ is proven similarly to Lemma 4.8:

Lemma 5.7. For all ground multisets of facts F, F' and F'' , all closed queries Q, Q_1, Q_2 , and ∇P . R in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$, and all closed conditions ϕ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{end}}$, the following statements hold:

1. If $I_Q(F) \rightarrow^! \Gamma$ then $\Gamma = \mathbf{a}(G)$ for some ground multiset of facts G .
2. Let f be a fact. If $I_f(F) \rightarrow^! \Gamma$ then $\Gamma = \mathbf{a}(f)$. If $I_{\emptyset}(F) \rightarrow^! \Gamma$ then $\Gamma = \mathbf{a}(\emptyset)$.

3. Let $F' \neq \emptyset$. In this case $I_{\phi \Rightarrow Q}(F) \rightarrow^! \mathbf{a}(F')$ iff $I_{\phi}(F) \rightarrow^! \mathbf{s}(\mathbf{t})$ and $I_Q(F) \rightarrow^! \mathbf{a}(F')$.
4. $I_{\phi \Rightarrow Q}(F) \rightarrow^! \mathbf{a}(\emptyset)$ iff either (non-exclusively) $I_{\phi}(F) \rightarrow^! \mathbf{s}(\mathbf{f})$ or $I_Q(F) \rightarrow^! \mathbf{a}(\emptyset)$.
5. $I_{Q_1 \triangleright Q_2}(F) \rightarrow^! \mathbf{a}(F')$ iff $I_{Q_1}(F) \rightarrow^! \mathbf{a}(F_1)$ and $I_{Q_2}(F) \rightarrow^! \mathbf{a}(F_2)$ for some multisets F_1 and F_2 such that $F' = F_1 \circ F_2$.
6. $I_{\nabla P.R}(F) \rightarrow^! \mathbf{a}(F')$ iff there exist lists of ground multisets of facts $F_0, \dots, F_n, G_0, \dots, G_{n-1}$, and H_0, \dots, H_{n-1} , and a sequence of substitutions $\sigma_0, \sigma_1, \dots, \sigma_{n-1}$ such that
 - (a) $F_0 = F, F_{i+1} = G_i \circ \sigma_i(P_i)$, and $F_i = G_i \circ \sigma_i(P_i \circ P_?)$, for all $i \in \{0, \dots, n-1\}$,
 - (b) $I_{\sigma_i(R)}(F) \rightarrow^! \mathbf{a}(H_i)$, for all $i \in \{0, \dots, n-1\}$,
 - (c) there exists no substitution σ_n and multiset of facts G_n such that $F_n = G_n \circ \sigma_n(P_i \circ P_?)$,
 - (d) $F' = H_0 \circ H_1 \circ \dots \circ H_{n-1}$.

Lemma 5.8. The following logical equivalences hold between conditions in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$:

$$\begin{aligned} \emptyset \triangleright Q \equiv Q, \quad Q_1 \triangleright Q_2 \equiv Q_2 \triangleright Q_1, \quad Q_1 \triangleright (Q_2 \triangleright Q_3) \equiv (Q_1 \triangleright Q_2) \triangleright Q_3, \\ \perp \Rightarrow R \equiv \emptyset, \quad \neg \perp \Rightarrow R \equiv R, \quad \exists P. \emptyset \equiv \emptyset \end{aligned}$$

The next results show that non-confluence of queries makes some natural equivalences invalid:

Lemma 5.9. Let ϕ be a condition in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ and let Q_1, Q_2 be queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$. For all ground multisets of facts F, F' and all substitutions σ such that $\sigma(\phi), \sigma(Q_1)$ and $\sigma(Q_2)$ are closed, we have

$$I_{\sigma((\phi \Rightarrow Q_1) \triangleright (\phi \Rightarrow Q_2))}(F) \rightarrow^! \mathbf{a}(F') \quad \text{if} \quad I_{\sigma(\phi \Rightarrow (Q_1 \triangleright Q_2))}(F) \rightarrow^! \mathbf{a}(F'), \quad (17)$$

however, the inverse implication does not hold in general. If ϕ is deterministic (Definition 4.12), then

$$(\phi \Rightarrow Q_1) \triangleright (\phi \Rightarrow Q_2) \equiv \phi \Rightarrow (Q_1 \triangleright Q_2). \quad (18)$$

Proof:

To prove the implication (17) assume $I_{\sigma(\phi \Rightarrow (Q_1 \triangleright Q_2))}(F) \rightarrow^! \mathbf{a}(F')$. Either $F' \neq \emptyset$ or $F' = \emptyset$. In the first case, by Lemma 5.7, p. 3, our assumption is equivalent to $I_{\phi}(F) \rightarrow^! \mathbf{r}(\mathbf{t})$ and $I_{\sigma(Q_1 \triangleright Q_2)}(F) \rightarrow^! \mathbf{a}(F')$, the latter of which, in turn, is equivalent, by Lemma 5.7, p. 5, to $I_{\sigma(Q_1)}(F) \rightarrow^! \mathbf{a}(F_1)$ and $I_{\sigma(Q_2)}(F) \rightarrow^! \mathbf{a}(F_2)$ for some multisets F_1 and F_2 such that $F' = F_1 \circ F_2$. But this is equivalent, by Lemma 5.7, p. 3 and 5, to $I_{\sigma((\phi \Rightarrow Q_1) \triangleright (\phi \Rightarrow Q_2))}(F) \rightarrow^! \mathbf{a}(F')$. The case $F' = \emptyset$ is dealt with similarly.

To show that, in general, the inverse implication does not hold, consider $\phi := \exists[a \circ b(x)]?. \{x = 1\}$, $Q := \phi \Rightarrow (c \triangleright d)$, $Q' := (\phi \Rightarrow c) \triangleright (\phi \Rightarrow d)$, where $a, c, d : \rightarrow \text{Fact}$ and $b : \text{Nat} \rightarrow \text{Fact}$. Let $F := a \circ b(1) \circ b(2)$. Since $I_{\phi}(F) \rightarrow^! \mathbf{s}(B)$ for $B \in \{\mathbf{t}, \mathbf{f}\}$, $I_Q(F) \rightarrow^! \mathbf{a}(F')$ iff $F' \in \{\emptyset, c \circ d\}$, whereas $I_{Q'}(F) \rightarrow^! \mathbf{a}(F')$ iff $F' \in \{\emptyset, c, d, c \circ d\}$.

Assume that ϕ is deterministic. By Theorem 4.13, either $I_{\sigma(\phi)}(F) \rightarrow^! \mathbf{s}(\mathbf{t})$ or $I_{\sigma(\phi)}(F) \rightarrow^! \mathbf{s}(\mathbf{f})$, but not both. Let $R_1 := \sigma((\phi \Rightarrow Q_1) \triangleright (\phi \Rightarrow Q_2))$, $R_2 := \sigma(\phi \Rightarrow (Q_1 \triangleright Q_2))$. If $I_{\sigma(\phi)}(F) \rightarrow^! \mathbf{s}(\mathbf{f})$ then $I_{R_i}(F) \rightarrow \mathbf{a}(F')$ iff $F' = \emptyset$ for $i \in \{1, 2\}$. If $I_{\sigma(\phi)}(F) \rightarrow^! \mathbf{s}(\mathbf{t})$ then, for $i \in \{1, 2\}$, $I_{R_i}(F) \rightarrow \mathbf{a}(F')$ iff $F' = F_1 \circ F_2$ for some multisets F_1 and F_2 such that $I_{Q_j}(F) \rightarrow \mathbf{a}(F_j)$, $j \in \{1, 2\}$. \square

Example 5.10. In general, equivalence $\nabla P . (Q_1 \triangleright Q_2) \equiv (\nabla P . Q_1) \triangleright (\nabla P . Q_2)$ is not valid. Indeed, let $a : \rightarrow \text{Fact}$ and $b, c : \text{Nat} \rightarrow \text{Fact}$, and let $P := [a \circ b(x)]?$, $Q := \nabla P . (b(x) \triangleright c(x))$, $Q' := (\nabla P . b(x)) \triangleright (\nabla P . c(x))$. Suppose that $F := a \circ b(1) \circ b(2)$. Then $I_Q(F) \rightarrow^! \tau(F')$ iff $F' \in \{b(i) \circ c(i) \mid i \in \{1, 2\}\}$. However, $I_{Q'}(F) \rightarrow^! \tau(F')$ iff $F' \in \{b(i) \circ c(j) \mid i, j \in \{1, 2\}\}$.

Here we define a class of queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ which evaluate to unique answers:

Definition 5.11. A query Q in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ is called *deterministic* if all quantification patterns in Q (including those inside conditions) contain only single facts with unique matching property.

Theorem 5.12. Let Q be a deterministic query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$. Then $\mathcal{R}_{\Sigma, \mathcal{D}}(Q)$ is confluent, and, in particular, given a ground multiset of facts F , there is a unique multiset of facts F' such that $I_Q(F) \rightarrow^! \alpha(F')$.

Proof:

As semiconfluence implies confluence, to show confluence at $t : \text{State}^q$ it suffices to prove that if $t' \leftarrow t \rightarrow^+ t''$ for some t' and t'' , then there exists s such that $t' \rightarrow^* s \leftarrow t''$. Semiconfluence is immediate at irreducible terms $\alpha(F')$, as well as terms $\{F, F', \emptyset\}^q$ which can only rewrite to $\alpha(F')$. Let $t := \{F, F', SA\}^q$ where $S : \text{Stk}^q$, and $A : \text{Frm}^q$. If there exists a unique multiset of facts K such that every rewrite sequence starting with t either (1) contains $h := \{F, F' \circ K, S\}^q$ or (2) it can be extended to reach h , then semiconfluence holds at t . Indeed, in this case, either $s = t''$ witnesses the semiconfluence (if (1) holds for $t \rightarrow^+ t''$) or $s = h$ does (if (2) holds for $t \rightarrow^+ t''$). It remains to prove the existence of the unique multiset K . We argue by induction on the structure of formulas indexing the frame A on the top of the stack. Most cases are dealt by trivial application of Lemmas 5.7 and 5.2. For frames related to conditionals $_ \Rightarrow _$ we have to also use Lemma 4.13. The only non-trivial part of the proof concerns frames A of the form $[F'' \mid \vec{a}]_{\nabla P, R}^{\vec{v}|q}$.

Under the theorem's assumption, $P = [f]?$, where f is a fact with unique matching property (Definition 4.11). Let \vec{w} be a sequence of all variables in $\text{Var}(f) \setminus \{\vec{v}\}$. If there is no fact f' in F' such that f' matches $\{\vec{a}/\vec{v}\}(f)$, then necessarily $K = \emptyset$. Otherwise $F' = G \circ f_1 \circ \dots \circ f_n$ for some $n > 0$, where (1) for all $i \in \{1, \dots, n\}$ there exists a unique substitution $\sigma_i = \{\vec{a}/\vec{v}, \vec{b}^i/\vec{w}\}$ such that $f_i = \sigma_i(f)$, (2) there is no fact f' in G such that f' matches $\{\vec{a}/\vec{v}\}(f)$. In this case, necessarily, by Lemma 5.7, p. 6, $K = K_1 \circ \dots \circ K_n$, where, for all $i \in \{1, \dots, n\}$, K_i is the unique multiset of facts (uniqueness and existence follows from inductive assumption) such that $I_{\sigma_i(R)}(F) \rightarrow^! \alpha(K_i)$. \square

There is a useful relationship between queries and conditions:

Lemma 5.13. Let $r : s_1 \dots s_n \rightarrow \text{Fact}$, let Q be a query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$, and let $\vec{x} := x_1, \dots, x_n$ be a list of n distinct variables such that $\{\vec{x}\} \cap \text{Var}(Q) = \emptyset$ and $x_i : s_i$ for $i \in \{1, \dots, n\}$. Then, there exists a condition $\phi_Q^{\vec{x}|r}$ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{cnd}}$ such that, for all ground multisets of facts F , and for all ground substitutions $\sigma := \{\vec{t}/\vec{x}, \vec{a}/\vec{v}\}$ such that $\{\vec{a}/\vec{v}\}(Q)$ is closed, $\sigma(\phi_Q^{\vec{x}|r})$ is closed, and, moreover,

$$\begin{aligned} I_{\sigma(\phi_Q^{\vec{x}|r})}(F) \rightarrow^! \mathfrak{s}(\mathbf{t}) &\quad \text{iff} \quad \exists F' . (I_Q(F) \rightarrow^! \alpha(F' \circ r(\vec{t}))), \\ I_{\sigma(\phi_Q^{\vec{x}|r})}(F) \rightarrow^! \mathfrak{s}(\mathbf{f}) &\quad \text{iff} \quad \exists F' . (r(\vec{t}) \notin F' \wedge I_Q(F) \rightarrow^! \alpha(F')), \end{aligned} \tag{19}$$

i.e., $\sigma(\phi_Q^{\vec{x}|r})$ evaluates to **t** (resp. **f**) on F if and only if there is some F' returned by Q when evaluated against F , such that $r(\vec{t}) \in F'$ (resp. $r(\vec{t}) \notin F'$). Moreover, if Q is deterministic, then so is $\phi_Q^{\vec{x}|r}$.

Proof:

We define $\phi_Q^{\vec{x}|r}$ by recursion on the structure of a query Q :

$$\phi_{\emptyset}^{\vec{x}|r} = \perp, \quad \phi_{h(\vec{t})}^{\vec{x}|r} = \{r(\vec{x}) = h(\vec{t})\}, \quad \phi_{Q_1 \triangleright Q_2}^{\vec{x}|r} = \phi_{Q_1}^{\vec{x}|r} \vee \phi_{Q_2}^{\vec{x}|r}, \quad \phi_{\psi \Rightarrow R}^{\vec{x}|r} = \psi \wedge \phi_R^{\vec{x}|r}, \quad \phi_{\nabla P.R}^{\vec{x}|r} = \exists P. \phi_R^{\vec{x}|r}.$$

The easy if laborious proof that $\phi_Q^{\vec{x}|r}$ really satisfies all the conditions in the statement is left to the reader. \square

Example 5.14. Consider the following query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ (where x and y are distinct variables):

$$Q := \nabla[f(x, y)]?. ((\{x = y\} \Rightarrow r(x)) \triangleright r(s(x))).$$

Thus, for any fact of the form $f(x, y)$ in the database, Q will output $r(s(x))$, and, if $x = y$, also $r(x)$. Using the recursive formula from the proof of Lemma 5.13 we easily see that

$$\phi_Q^{z|r} := \exists[f(x, y)]?. ((\{x = y\} \wedge \{r(z) = r(x)\}) \vee \{r(z) = r(s(x))\}).$$

The next result shows that $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ can emulate relational algebra.

Theorem 5.15. Denote by $\text{RelAlg}(\mathcal{S}, \mathcal{D})$ the relational algebra over the relational schema \mathcal{S} and the domain \mathcal{D} of atomic values (which we silently identify with its algebraic representation, where types are sorts and predicates are represented with equationally defined operators into the `Bool` sort). Furthermore, assume that for each relational algebra expression R in $\text{RelAlg}(\mathcal{S}, \mathcal{D})$ a function symbol $\mathcal{R}_R : s_1 \dots s_n \rightarrow \text{Fact}$ is in Σ_F , where s_i is the sort (domain) of the i -th column of R and n is the arity of R . For any relational database I with schema \mathcal{S} , let $\overline{tr}(I)$ be a multiset corresponding to the database I . More precisely,

$$\overline{tr}(I) := \circ\{\mathcal{R}_r(\vec{t}) \mid r \text{ is a relation symbol in } \mathcal{S} \text{ and } (\vec{t}) \in r^I\},$$

where r^I is the set of tuples of r in I . Then, for all formulas R in $\text{RelAlg}(\mathcal{S}, \mathcal{D})$, there exists a closed query $tr(R)$ in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ such that for all relational databases I with schema \mathcal{S} ,

$$\exists F. (I_{tr(R)}(\overline{tr}(I)) \rightarrow^! \mathbf{a}(F \circ \mathcal{R}_R(\vec{t}))) \quad \text{iff} \quad (\vec{t}) \in \text{eval}_I(R), \quad (20)$$

where $\text{eval}_I(R)$ is the set of tuples obtained by evaluation of relational query R against the relational database I . Furthermore, for any relational expression R , $tr(R)$ is deterministic.

Proof:

To prove the theorem we define $tr(R)$ by recursion on the structure of R . We consider relational algebra expressions to be constructed with base relations, projections, selections, set unions, Cartesian products, and set differences. Other well known relational operators such as joins can be defined in

terms of basic operators mentioned above. Attribute renaming is not relevant in our case, since we represent relations with positional arguments only. We denote set unions, Cartesian products, and set differences with the usual mathematical notation (e.g., $R \cup S$, $R \times S$ and $R \setminus S$, respectively). Notation for projections and selections is less standardised (and needs to be adapted to relations with positional arguments). We denote by $\pi_{i_1, \dots, i_k}(R)$ the projection of R onto i_1 'th, i_2 'th, \dots , and i_k 'th “leg” (i.e., on a single tuple, $\pi_{i_1, \dots, i_k}((t_1, t_2, \dots, t_n)) := (t_{i_1}, t_{i_2}, \dots, t_{i_k})$). We denote by $\sigma_\phi(R)$ the selection with condition ϕ applied to R (i.e., it returns those tuples of R which satisfy ϕ). If R is n -ary, then we will assume that in ϕ expression $\$i$ corresponds to the i -th column of R , for $i \in \{1, \dots, n\}$.

Let r be a base relation of arity k (for simplicity we omit the typing information), and let \vec{x} be a list of k distinct variables. Then $tr(r) := \nabla[\mathcal{R}_r(\vec{x})]_? \cdot \mathcal{R}_r(\vec{x})$. We now consider selected relational operators:

1. Let R be a relational formula defining an n -ary relation, and let i_1, \dots, i_k be a subsequence of $1, \dots, n$. We define $tr(\pi_{i_1, \dots, i_k}(R))$ to be $tr(R)$ with each subquery of the form $\mathcal{R}_R(t_1, \dots, t_n)$ replaced with $\mathcal{R}_{\pi_{i_1, \dots, i_k}(R)}(t_{i_1}, \dots, t_{i_k})$.
2. Let R be a relational formula of arity k , and let ϕ be a term of sort `Bool` representing condition on rows of R (where in ϕ special variable $\$i$, $i \in \{1, \dots, n\}$ corresponds to the i -th “attribute” of R). We define $tr(\sigma_\phi(R))$ to be $tr(R)$ with each subquery of the form $\mathcal{R}_R(\vec{t})$ replaced with $\{\{\vec{t}/\$\}(\phi)\} \Rightarrow \mathcal{R}_{\sigma_\phi(R)}(\vec{t})$.
3. $tr(R_1 \cup R_2) := tr(R_1) \triangleright tr(R_2)$.
4. Let R and S be relational formulas. Let $tr(S)' := \sigma(tr(S))$ for some renaming σ such that $Var(tr(R)) \cap Var(tr(S)') = \emptyset$ ($tr(S) \equiv tr(S)'$ by Lemma 5.6 since $tr(S)$ is closed). Further, let $\alpha(\vec{t})$ be $tr(S)'$ with each subquery of the form $\mathcal{R}_S(\vec{s})$ replaced with $\mathcal{R}_{R \times S}(\vec{t}, \vec{s})$. Then we define $tr(R \times S)$ to be $tr(R)$ with each subquery of the form $\mathcal{R}_R(\vec{t})$ replaced with $\alpha(\vec{t})$.
5. Let R and S be relational formulas of arity k . Let $tr(S)'$ be like in the previous point. Let \vec{x} be a list of k distinct variables such that $\{\vec{x}\} \cap (Var(tr(R)) \cup Var(tr(S)')) = \emptyset$. We define $tr(R \setminus S)$ to be $tr(R)$ with each subquery of the form $\mathcal{R}_R(\vec{t})$ replaced with $\neg\{\vec{t}/\vec{x}\}(\phi_{tr(S)'}) \Rightarrow \mathcal{R}_{R \setminus S}(\vec{t})$ (see Lemma 5.13).

An easy induction on the structure of R shows that $tr(R)$ is closed and deterministic, and Equation (20) is satisfied (in the case of induction step for set difference we also use Lemma 5.13) \square

Remark 5.16. Relational queries R evaluate to sets of tuples. However, $tr(R)$ may evaluate to a multiset of facts — e.g., when evaluating unions duplicate facts are not removed.

Example 5.17. Let r and s be binary relations. Consider the following relational algebra expression:

$$R := \pi_{1,4}(\sigma_{\$2=\$3}(r \times s)).$$

Thus, $(x, y) \in R$ iff $(x, z) \in r$ and $(z, y) \in s$ for some z . Let represent R as a query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ using definition of $tr(-)$ from the proof of Theorem 5.15. First,

$$tr(r) = \nabla[\mathcal{R}_r(x_1, x_2)]_? \cdot \mathcal{R}_r(x_1, x_2), \quad tr(s) = \nabla[\mathcal{R}_s(y_1, y_2)]_? \cdot \mathcal{R}_s(y_1, y_2),$$

where x_1, x_2, y_1, y_2 are distinct variables. Then, by the point 4 in the proof

$$tr(r \times s) = \nabla[\mathcal{R}_r(x_1, x_2)]? \cdot \nabla[\mathcal{R}_s(y_1, y_2)]? \cdot \mathcal{R}_{r \times s}(x_1, x_2, y_1, y_2).$$

Finally, to deal with selection and projection we apply points 2 and 1, respectively, from the proof:

$$\begin{aligned} & tr(\pi_{1,4}(\sigma_{\mathbb{S}_2=\mathbb{S}_3}(r \times s))) \\ &= \nabla[\mathcal{R}_r(x_1, x_2)]? \cdot \nabla[\mathcal{R}_s(y_1, y_2)]? \cdot (\{x_2 = y_1\} \Rightarrow \mathcal{R}_{\pi_{1,4}(\sigma_{\mathbb{S}_2=\mathbb{S}_3}(r \times s))}(x_1, y_2)). \end{aligned}$$

6. Rewriting semantics of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$

We associate with a DML query Q in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ the rewriting system $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$. Terms rewritten with the rules of $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$ are of the form $\{F, F', F_n, S\}^d$, where $\{-, -, -, -\}^d : \text{FSet FSet FSet}^n \text{Stk}^d \rightarrow \text{State}^d$. F is the database of facts against which we issue the DML query. F changes during execution of the query while the facts are removed from it. A multiset F' , expanded during query execution, contains new facts to be added to the database. F_n is a multiset of fresh facts (see Section 2) from which fresh values are drawn. S is a stack of sort Stk^d which simulates structural recursion. Normal forms encapsulating a new database and new multiset of fresh facts after successful or, respectively, failing execution of a DML query, are constructed with $\mathfrak{n}, \mathfrak{f} : \text{FSet FSet}^n \rightarrow \text{State}^d$. We consider only terms t of sort State^d which satisfy the following *freshness condition*:

Definition 6.1. A term t of sort State^d satisfies the *freshness condition* if and only if $m > n$ for all positions κ, κ' in t such that $\kappa \neq \kappa'1$, $t|_{\kappa} = C_s(i_m^s)$ and $t|_{\kappa'} = i_n^s$.

Example 6.2. Consider the following term of State^d sort:

$$\mathfrak{n}(f(i_{10}^s) \circ f(i_3^s), C_s(i_7^s)).$$

It does not satisfy the freshness condition having as subterms both $C_s(i_7^s)$ and i_{10}^s with $10 > 7$. Our general assumption which justifies the freshness condition is that a fresh fact of the form $C_s(i_m^s)$ means that we never before used any value of the form i_n^s with $n > m$. Clearly, terms which do not satisfy freshness condition (like the term above) violate this assumption.

Terms of sort Stk^d are constructed from local computation frames of sort Frm^d , where $\text{Frm}^d < \text{Stk}^d$, using an associative binary operator $_ : \text{Stk}^d \text{Stk}^d \rightarrow \text{Stk}^d$ with identity \emptyset . Most constructors of frames are indexed by DML sub-queries R of Q , and lists of distinct variable names $\vec{v} := v_1, \dots, v_n$ of respective sorts s_1, \dots, s_n such that $\{\vec{v}\} \subseteq \text{Var}(Q)$ contains all free variables of R :

$$\begin{aligned} & [-, \dots, -]_R^{\vec{v}|d}, [-, \dots, -]_R^{\vec{v}|d, \downarrow} : s_1 \dots s_n \rightarrow \text{Frm}^d, \quad [-, \dots, -]_R^{\vec{v}|d} : s_1 \dots s_n \text{Stk}^c \rightarrow \text{Frm}^d, \\ & \checkmark : \rightarrow \text{Frm}^d, \quad [- | -, \dots, -]_{\nabla P, R}^{\vec{v}|d} : \text{FSet } s_1 \dots s_n \text{Bool} \rightarrow \text{Frm}^d, \\ & [- | -, \dots, - | -, -]_{\nabla P, R}^{\vec{v}|d} : \text{FSet } s_1 \dots s_n \text{Bool FSet} \rightarrow \text{Frm}^d. \end{aligned}$$

As \vec{v} can be empty, the above signature templates include $\llbracket \vec{v} \rrbracket_R^d, \llbracket \vec{v} \rrbracket_{R,\downarrow}^d \rightarrow \text{Frm}^d$, etc. A constant \surd marks successful branches of computation, i.e., those which created either new facts or $\surd : \text{DQy}$. Marking such branches is necessary as facts deleted by unsuccessful branches are restored as soon as the branch finishes. Frames of the form $[\vec{a}]_R^{\vec{v}^d}, [\vec{a}]_{R,\downarrow}^{\vec{v}^d}, [\vec{a}|S]_R^{\vec{v}^d}, [F'|\vec{a}|B]_R^{\vec{v}^d}$, or $[F'|\vec{a}|B, F'']_R^{\vec{v}^d}$ are called (R, σ) -frames, where $\sigma := \{\vec{a}/\vec{v}\}$ is the current substitution. They are related to evaluation of $\sigma(R)$. Marked frames $[\vec{a}]_{R,\downarrow}^{\vec{v}^d}$ occur in the execution of “unions” $\triangleright \triangleright _$. Conditional frames $[\vec{a}|S]_R^{\vec{v}^d}$ are used in execution of conditionals $\phi \Rightarrow R$, where $S : \text{Stk}^c$ represents evaluation of ϕ . Iterator frames $[F'|\vec{a}|B]_{\nabla P.R}^{\vec{v}^d}$ and $[F'|\vec{a}|B, F'']_{\nabla P.R}^{\vec{v}^d}$ represent iterative execution of $\sigma(\nabla P.R)$. Multiset F' , called iterator state, contains facts available for matching with $P_1 \circ P_2 \circ P_0$ (cf. Remark 2.3). Iteration status B is equal to \mathbf{t} iff the iteration already generated either new facts or \surd (i.e., if the branch related to $\sigma(\nabla P.R)$ was successful). “Tentative” iterator frames $[F'|\vec{a}|B, F'']_{\nabla P.R}^{\vec{v}^d}$ store multiset F'' of facts deleted from the database in the present step, so that they can be restored if the step is unsuccessful. Given a database F , in order to execute a closed DML query Q we rewrite a state term $\text{I}_Q(F, F_{\mathbf{n}}) := \{F, \emptyset, F_{\mathbf{n}}, \llbracket Q \rrbracket^d\}^d$ until a normal form $\mathbf{n}(F', F'_{\mathbf{n}})$ or $\mathbf{f}(F', F'_{\mathbf{n}})$ is reached, indicating that a successful or, respectively, unsuccessful execution of Q in the database F yielded a new database F' , and a new multiset of fresh facts $F'_{\mathbf{n}}$. Now we are ready to define rule schemas of $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$.

Two consecutive occurrences of \surd are collapsed, and (\surd, σ) -frames are replaced with \surd :

$$\lambda_{\text{col}} : \{F, F', F_{\mathbf{n}}, S \surd \surd\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S \surd\}^d, \quad \lambda_{\surd} : \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{\surd}^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S \surd\}^d. \quad (21)$$

An (\emptyset, σ) -frame is removed from the stack. An (f, σ) -frame, where f is a fact and σ is the current substitution, is replaced by \surd and $\sigma(f)$ is added to F' (since $\text{Var}(f) \subseteq \{\vec{v}\}$, $\sigma(f)$ is closed):

$$\lambda_{\emptyset} : \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{\emptyset}^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S\}^q, \quad \lambda_{\text{fact}} : \{F, F', F_{\mathbf{n}}, S[\vec{v}]_f^{\vec{v}^d}\}^d \Rightarrow \{F, F' \circ f, F_{\mathbf{n}}, S \surd\}^q. \quad (22)$$

An $(R_1 \triangleright R_2, \sigma)$ -frame is split into the (R_1, σ) -frame and the (R_2, σ) -frame. The (R_2, σ) -frame is marked with \downarrow so that the evaluation of $\sigma(R_1 \triangleright R_2)$ can be marked as successful when at least one of the branches is successful. When both branches are successful, this can produce two consecutive \surd constants on the stack which are then collapsed using λ_{col} rule in Equation (21).

$$\begin{aligned} \lambda_{\triangleright}^{\text{unf}} &: \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{R_1 \triangleright R_2}^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{R_2}^{\vec{v}^d, \downarrow} [\vec{v}]_{R_1}^{\vec{v}^d}\}^d, \\ \lambda_{\triangleright; \surd}^{\text{fld}} &: \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{R_2}^{\vec{v}^d, \downarrow} \surd\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S \surd [\vec{v}]_{R_2}^{\vec{v}^d}\}^d, \\ \lambda_{\triangleright; \emptyset}^{\text{fld}} &: \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{R_2}^{\vec{v}^d, \downarrow}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{R_2}^{\vec{v}^d}\}^d, \end{aligned} \quad (23)$$

Rule schemas for execution of conditionals $\phi \Rightarrow Q$ are similar to those in Equations (12), (13):

$$\begin{aligned} \lambda_{\text{cond}}^{\text{unf}} &: \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{\phi \Rightarrow R}^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S[\vec{v} \mid [\vec{v}]_{\phi}^{\vec{v}^c}]_{R}^{\vec{v}^d}\}^d, \\ \lambda_{\text{cond}; \mathbf{f}}^{\text{unf}} &: \{F, F', F_{\mathbf{n}}, S[\vec{v} \mid \mathbf{t}(\mathbf{f})]_R^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S\}^d, \\ \lambda_{\text{cond}; \mathbf{t}}^{\text{unf}} &: \{F, F', F_{\mathbf{n}}, S[\vec{v} \mid \mathbf{t}(\mathbf{t})]_R^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S[\vec{v}]_R^{\vec{v}^d}\}^d, \\ \lambda^d &: \{F, F', F_{\mathbf{n}}, S[\vec{v} \mid S']_R^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S[\vec{v} \mid S'']_R^{\vec{v}^d}\}^d \text{ if } C \\ &\text{for all } \lambda : \{F, S'\}^c \Rightarrow \{F, S''\}^c \text{ if } C \text{ in } \mathcal{R}_{\Sigma, \mathcal{D}}^{\text{cnd}}(\phi). \end{aligned} \quad (24)$$

Evaluation of a ∇_{\dots} subquery is initialized with the whole database available for matching. Iteration status is \mathbf{f} since evaluation is not successful yet.

$$\lambda_{\nabla}^{\text{init}} : \{F, F', F_{\mathbf{n}}, S[\vec{v}]_{\nabla P.R}^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S[F' \mid \vec{v} \mid \mathbf{f}]_{\nabla P.R}^{\vec{v}^d}\}^d. \quad (25)$$

Let \vec{w} be a sequence of all the distinct variables in $\text{Var}(P) \setminus \{\vec{v}\}$. Let σ be the current substitution. Rule $\lambda_{\nabla}^{\text{unf}}$ pushes onto the stack a (σ', R) -frame, where $\sigma' = \sigma \cup \{\vec{b}/\vec{w}\}$ is defined by matching $F'' \circ \sigma(P_! \circ P_? \circ P_0)$ with iterator state and $P_{\mathbf{n}}$ with the multiset of fresh facts $F_{\mathbf{n}}$. It also removes $\sigma'(P_? \circ P_0)$ from the iterator state and $\sigma'(P_0)$ from the database state, and, finally, it updates fresh facts using v defined in Equation (1). Removed facts $\sigma'(P_0)$ are stored in the tentative iterator frame:

$$\begin{aligned} \lambda_{\nabla}^{\text{unf}} : & \{F \circ P_! \circ P_? \circ P_0, F', F_{\mathbf{n}} \circ P_{\mathbf{n}}, S[F'' \circ P_! \circ P_? \circ P_0 \mid \vec{v} \mid B]_{\nabla P.R}^{\vec{v}^d}\}^d \\ & \Rightarrow \{F \circ P_! \circ P_?, F', F_{\mathbf{n}} \circ v(P_{\mathbf{n}}), S[F'' \circ P_! \mid \vec{v} \mid B, P_0]_{\nabla P.R}^{\vec{v}^d}[\vec{v}, \vec{w}]_R^{\vec{v}, \vec{w}^d}\}^d, \end{aligned} \quad (26)$$

If execution of $\sigma'(R)$ proves unsuccessful, removed facts can be restored both to iterator state and database. Otherwise, we discard them and set the iteration status to \mathbf{t} :

$$\begin{aligned} \lambda_{\nabla; \emptyset}^{\text{fd}} : & \{F, F', F_{\mathbf{n}}, S[F'' \mid \vec{v} \mid B, F_0]_{\nabla P.R}^{\vec{v}^d}\}^d \Rightarrow \{F \circ F_0, F', F_{\mathbf{n}}, S[F'' \circ F_0 \mid \vec{v} \mid B]_{\nabla P.R}^{\vec{v}^d}\}^d, \\ \lambda_{\nabla; \surd}^{\text{fd}} : & \{F, F', F_{\mathbf{n}}, S[F'' \mid \vec{v} \mid B, F_0]_{\nabla P.R}^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S[F'' \mid \vec{v} \mid \mathbf{t}]_{\nabla P.R}^{\vec{v}^d}\}^d. \end{aligned} \quad (27)$$

We keep applying $\lambda_{\nabla}^{\text{unf}}$ until we cannot match $\sigma(P_! \circ P_? \circ P_0)$ with iterator state. Then we replace the iterator frame with $\delta(B)$ where B is the iteration status, $\delta(\mathbf{t}) = \surd$, and $\delta(\mathbf{f}) = \emptyset$. To prevent premature application, rule schema $\lambda_{\nabla}^{\text{end}}$ is conditional, where the condition uses functions $\mu_{P, \vec{v}} : \text{Pat } s_1 \dots s_n \rightarrow \text{Yes?}$ defined for each $\vec{v} \subseteq \text{Var}(Q)$ and pattern P occurring in Q with the single equation $\mu_{P, \vec{v}}(F \circ P_! \circ P_? \circ P_0, \vec{v}) = \text{yes}$ (cf. Equation (8)):

$$\lambda_{\nabla}^{\text{end}} : \{F, F', F_{\mathbf{n}}, S[F'' \mid \vec{v} \mid B]_{\nabla P.R}^{\vec{v}^d}\}^d \Rightarrow \{F, F', F_{\mathbf{n}}, S\delta(B)\}^d \text{ if } (\mu_{P, \vec{v}}(F'', \vec{v}) = \text{yes}) = \mathbf{f}. \quad (28)$$

The last two rules reduce the State^d -terms with an empty stack or stack containing only the \surd constant into a term constructed with \mathbf{n} or \mathbf{f} , respectively:

$$\lambda_{\text{dml}}^{\surd} : \{F, F', F_{\mathbf{n}}, \surd\} \Rightarrow \mathbf{n}(F \circ F', F_{\mathbf{n}}), \quad \lambda_{\text{dml}}^{\emptyset} : \{F, F', F_{\mathbf{n}}, \emptyset\} \Rightarrow \mathbf{f}(F \circ F', F_{\mathbf{n}}). \quad (29)$$

Theorem 6.3. $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$ is a terminating rewriting system.

Proof:

The proof is similar to the proof of Theorem 4.2. Only subqueries of the form $\nabla P . R$, where P is semiterminating but not terminating (i.e., $P_0 \neq \emptyset$, but $P_? = \emptyset$) require special care. In this case, the signature of ∇_{\dots} forces R to be success assured, i.e., R 's evaluation always succeeds, and hence removed facts matching P_0 are never restored. This ensures termination of $\nabla P . R$'s execution. \square

The following useful observation can be trivially verified by examining the rule schemas:

Lemma 6.4. Let R be a DML subquery of Q . Then, for all multisets of facts F, F', G, G' , multisets F_n, G_n of fresh facts, stacks S , lists of variables $\vec{v} = v_1, \dots, v_n$ and of values $\vec{a} = a_1, \dots, a_n$,

1. $\{F, F', F_n, S[\vec{a}]_R^{\vec{v}^d}\}^d \rightarrow_{\mathcal{R}_1}^* \{G, F' \circ G', G_n, S\sqrt{\ }^d \text{ iff } \{F, \emptyset, F_n, \square_{\sigma(R)}^d\}^d \rightarrow_{\mathcal{R}_2}^* \mathfrak{n}(G \circ G', G_n)$,
2. $\{F, F', F_n, S[\vec{a}]_R^{\vec{v}^d}\}^d \rightarrow_{\mathcal{R}_1}^* \{F, F', G_n, S\}^d \text{ iff } \{F, \emptyset, F_n, \square_{\sigma(R)}^d\}^d \rightarrow_{\mathcal{R}_2}^* \mathfrak{f}(F, G_n)$.

where $\sigma := \{\vec{a}/\vec{v}\}$, $\mathcal{R}_1 := \mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$, and $\mathcal{R}_2 := \mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(\sigma(R))$.

As in the case of pure queries, we consider two DML queries equivalent if and only if they can match their results. We should not, however, distinguish results differing only by the choice of fresh values. Let \mathcal{S}_n denote the set of nominal sorts in Σ_S . Let $\text{nom}(t)$ be the \mathcal{S}_n -sorted set of nominal values contained in term t . For any \mathcal{S}_n -sorted bijection $\alpha : X \rightarrow Y$ between sets of nominal values we denote by $\hat{\alpha}$ the natural extension of α to terms t such that $\text{nom}(t) \subseteq X$. More precisely, $\hat{\alpha}(x) = \alpha(x)$ if $x \in X$, $\hat{\alpha}(c) = c$ if c is a constant of non-nominal sort or a variable, and $\hat{\alpha}(f(t_1, \dots, t_n)) = f(\hat{\alpha}(t_1), \dots, \hat{\alpha}(t_n))$ if $f(t_1, \dots, t_n)$ is of non-nominal sort. With those notions we define equivalence on DML queries as follows:

Definition 6.5. Let Q_1 and Q_2 be two DML queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$. We say that Q_1 is logically equivalent to Q_2 , writing $Q_1 \equiv Q_2$, if and only if, for all ground substitutions σ such that $\sigma(Q_1)$ and $\sigma(Q_2)$ are closed, all ground multisets of facts F, G , all ground multisets of fresh facts F_n, G_n , and all $i \in \{1, 2\}$

1. if $I_{\sigma(Q_i)}(F, F_n) \rightarrow^! \mathfrak{n}(G, G_n)$ then there exist multisets of fresh facts F'_n, G'_n , multisets of facts F', G' , and an \mathcal{S}_n -sorted bijection $\alpha : \text{nom}(F) \cup \text{nom}(G) \rightarrow \text{nom}(F') \cup \text{nom}(G')$ such that $F' = \hat{\alpha}(F)$, $G' = \hat{\alpha}(G)$, and $I_{\sigma(Q_{3-i})}(F', F'_n) \rightarrow^! \mathfrak{n}(G', G'_n)$.
2. if $I_{\sigma(Q_i)}(F, F_n) \rightarrow^! \mathfrak{f}(F, G_n)$ then there exist multisets of fresh facts F'_n, G'_n , a multiset of facts F' , and an \mathcal{S}_n -sorted bijection $\alpha : \text{nom}(F) \rightarrow \text{nom}(F')$ such that $F' = \hat{\alpha}(F)$ and $I_{\sigma(Q_{3-i})}(F', F'_n) \rightarrow^! \mathfrak{f}(F', G'_n)$.

The following result is an immediate consequence of Lemma 6.4:

Lemma 6.6. Logical equivalence on queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ is an equivalence relation and a congruence, i.e., if κ is a position in a DML query Q such that $Q|_{\kappa}$ is a DML query, and $R \equiv Q|_{\kappa}$, then $Q \equiv Q[R]_{\kappa}$.

Lemma 6.7. For any closed DML query Q in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$, and any renaming σ , $Q \equiv \sigma(Q)$.

Lemma 6.8, proven similarly to Lemma 4.8, clarifies elements of rewriting semantics of DML queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ (we leave ∇_{\dots} evaluation where we are better off with the rewriting definition).

Lemma 6.8. For all ground multisets of facts F, G and of fresh facts F_n, G_n , as well as closed DML queries Q, Q_1, Q_2 and all sentences ϕ , the following holds:

1. If $I_Q(F, F_n) \rightarrow^! \Gamma$ then $\Gamma = \mathfrak{n}(G, G_n)$ or $\Gamma = \mathfrak{f}(F, G_n)$ for some ground multiset of facts G and ground multiset of fresh facts G_n . If $I_Q(F, F_n) \rightarrow^! \mathfrak{f}(G, G_n)$ then $F = G$.
2. If $I_f(F, F_n) \rightarrow^! \Gamma$ then $\Gamma = \mathfrak{n}(F \circ f, F_n)$. If $I_{\emptyset}(F, F_n) \rightarrow^! \Gamma$ then $\Gamma = \mathfrak{f}(F, F_n)$.

3. $I_{\phi \Rightarrow Q}(F, F_{\mathbf{n}}) \rightarrow^! \mathbf{n}(G, G_{\mathbf{n}})$ iff $I_{\phi}(F) \rightarrow^! \mathbf{r}(\mathbf{t})$ and $I_Q(F, F_{\mathbf{n}}) \rightarrow^! \mathbf{n}(G, G_{\mathbf{n}})$.
4. $I_{\phi \Rightarrow Q}(F, F_{\mathbf{n}}) \rightarrow^! \mathbf{f}(F, G_{\mathbf{n}})$ iff $I_{\phi}(F) \rightarrow^! \mathbf{r}(\mathbf{f})$ or $I_Q(F, F_{\mathbf{n}}) \rightarrow^! \mathbf{f}(F, G_{\mathbf{n}})$.
5. $I_{Q_1 \triangleright Q_2}(F, F_{\mathbf{n}}) \rightarrow^! \mathbf{f}(F, G_{\mathbf{n}})$ iff there exists a multiset of fresh facts $G'_{\mathbf{n}}$ such that $I_{Q_1}(F, F_{\mathbf{n}}) \rightarrow^! \mathbf{f}(F, G'_{\mathbf{n}})$ and $I_{Q_2}(F, G'_{\mathbf{n}}) \rightarrow^! \mathbf{f}(F, G_{\mathbf{n}})$.
6. $I_{Q_1 \triangleright Q_2}(F, F_{\mathbf{n}}) \rightarrow^! \mathbf{n}(G, G_{\mathbf{n}})$ iff, for some ground multisets of facts F', F'', G', G'' such that $G = G'' \circ F' \circ F''$, a ground multiset of fresh facts $G'_{\mathbf{n}}$, and stacks $S, S' \in \{\emptyset, \sqrt{}\}$ where $S = \sqrt{}$ or $S' = \sqrt{}$, we have $I_{Q_1}(F, F_{\mathbf{n}}) \rightarrow^* \{G', F', G'_{\mathbf{n}}, S\}^d$ and $I_{Q_2}(G', G'_{\mathbf{n}}) \rightarrow^* \{G'', F'', G_{\mathbf{n}}, S'\}^d$.

Lemma 6.9. The following logical equivalences hold between queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$:

$$\begin{aligned} \emptyset \triangleright Q \equiv Q, \quad Q \triangleright \emptyset \equiv Q, \quad Q_1 \triangleright (Q_2 \triangleright Q_3) \equiv (Q_1 \triangleright Q_2) \triangleright Q_3, \\ \perp \Rightarrow R \equiv \emptyset, \quad \neg \perp \Rightarrow R \equiv R, \quad \exists P. \emptyset \equiv \emptyset \end{aligned}$$

Lemma 6.9 is very similar to Lemma 5.8, except that $_ \triangleright _$ is not commutative. $Q_1 \triangleright Q_2$ may be non-equivalent with $Q_2 \triangleright Q_1$ if, say, Q_1 deletes a fact which is referred to in some pattern in Q_2 .

We need to generalize the notion of confluence, lest rewriting paths leading to terms differing only by distinct choices of fresh values (as in the next example) are to be considered non-convergent.

Example 6.10. Let \mathbf{I} be a nominal sort. Let $r : \text{Nat} \rightarrow \text{Fact}$, $s : \mathbf{I} \text{ Nat} \rightarrow \text{Fact}$. Consider DML query $Q := \nabla[C_{\mathbf{I}}(x)]_{\mathbf{n}} \circ [r(y)]_{?} . s(x, y)$, and let $F := r(1) \circ r(2)$. Then

$$\begin{aligned} I_Q(F, C_{\mathbf{I}}(\iota_0^{\mathbf{I}})) &\rightarrow^* \{F, \emptyset, C_{\mathbf{I}}(\iota_0^{\mathbf{I}}), [F \parallel \mathbf{f}]_Q^d\}^d \xrightarrow{\lambda_{\nabla}^{\text{unf}}} \{F, \emptyset, C_{\mathbf{I}}(\iota_1^{\mathbf{I}}), [r(2) \parallel \mathbf{f}, \emptyset]_Q^d [\iota_0^{\mathbf{I}}, 1]_{s(x,y)}^{x,y/d}\}^d \\ &\rightarrow^* \{F, s(\iota_0^{\mathbf{I}}, 1), C_{\mathbf{I}}(\iota_1^{\mathbf{I}}), [r(2) \parallel \mathbf{t}]_Q^d\}^d \xrightarrow{\lambda_{\nabla}^{\text{unf}}} \{F, S(\iota_0^{\mathbf{I}}, 1), C_{\mathbf{I}}(\iota_2^{\mathbf{I}}), [\emptyset \parallel \mathbf{t}]_Q^d [\iota_1^{\mathbf{I}}, 2]_{s(x,y)}^{x,y/d}\}^d \\ &\rightarrow^* \mathbf{n}(F \circ s(\iota_0^{\mathbf{I}}, 1) \circ s(\iota_1^{\mathbf{I}}, 2), C_{\mathbf{I}}(\iota_2^{\mathbf{I}})), \end{aligned}$$

and, if we match $r(1)$ and $r(2)$ in reverse order in applications of $\lambda_{\nabla}^{\text{unf}}$ rule, then

$$I_Q(F, C_{\mathbf{I}}(\iota_0^{\mathbf{I}})) \rightarrow^* \mathbf{n}(F \circ s(\iota_0^{\mathbf{I}}, 2) \circ s(\iota_1^{\mathbf{I}}, 1), C_{\mathbf{I}}(\iota_2^{\mathbf{I}})).$$

Here we define an equivalence relation on terms of sort State^d which is a bisimulation:

Definition 6.11. We say that term t_1 is nominally equivalent to term t_2 , in which case we write $t_1 \equiv_{\mathbf{n}} t_2$, if and only if there exists a bijection $\alpha : \text{nom}(t_1) \rightarrow \text{nom}(t_2)$ such that $\hat{\alpha}(t_1) = t_2$.

Lemma 6.12. Nominal equivalence is an equivalence relation. When restricted to State^d terms satisfying the freshness condition (Definition 6.1), it is also a bisimulation on $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$, for all Q .

We leave an easy proof of Lemma 6.12 to the reader. The restriction to terms satisfying the freshness condition is necessary for nominal equivalence being a bisimulation, as demonstrated below:

Example 6.13. Let I be a nominal sort. Let $r : I \rightarrow \text{Fact}$, $s : I \ I \rightarrow \text{Fact}$. Define $F := r(i_1^I) \circ r(i_2^I)$. Consider a DML query $Q := \nabla[C_I(x)]_{\mathbf{n}} \circ [r(y)]_? \cdot (\{x = y\} \Rightarrow s(x, y))$. Let $t_1 := I_Q(F, C_I(i_0^I))$ and $t_2 := I_Q(F, C_I(i_3^I))$. Term t_1 does not satisfy the freshness condition (Definition 6.1). It is immediate that $t_1 \equiv_{\mathbf{n}} t_2$, $t_1 \rightarrow^! t_3$, where $t_3 := \mathbf{n}(s(i_1^I, 1) \circ F, C_I(i_2^I))$, but the only normal form of t_2 is $t_4 := \mathbf{f}(F, C_I(i_5^I))$ and $t_3 \not\equiv_{\mathbf{n}} t_4$.

We now define a class of queries Q for which $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$ is confluent modulo nominal equivalence.

Definition 6.14. Let Q be a DML query. We say that Q has *no deletion conflicts* if and only if for each DML subquery $\nabla P \cdot R$ of Q , and any subterm $f : \text{Fact}$ of R (resp. P) occurring inside $[-]_0$, P (resp. R) has no subterm f' occurring inside $[-]_0$, $[-]_!$ or $[-]_?$ such that f and f' are unifiable.

Definition 6.15. Let Q be a DML query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$. Q is called *deterministic* if it has no deletion conflicts and all quantification patterns in Q (including those inside subterms which are conditions) contain only single facts with unique matching property (but may contain any number of fresh facts).

In Example 4.14 we shown why multiple facts in patterns lead to non-confluence, and as a result, to non-deterministic evaluation of conditions (and queries). However, we have not previously considered deletion conflicts (as they are specific to DML queries). The following example shows why deletion conflicts can prevent confluence:

Example 6.16. Consider the following DML query:

$$Q := \nabla[f(x)]_0 \cdot (\sqrt{\triangleright} (\nabla[f(1)]_? \cdot h(x))). \quad (30)$$

Observe that all quantification patterns in Q consist of single facts, but Q does have deletion conflicts (facts in both patterns are unifiable, and one of them is $[-]_0$ -pattern which has the second one in its scope). Since the first pattern ($[f(x)]_0$) is semi-terminating but not terminating, to ensure that the DML query in its scope is success assured it is of the form $\sqrt{\triangleright} _$.

Now, let $F := f(1) \circ f(2)$. It is easy to see that executing Q against F removes from F both f -facts and either adds a single fact $h(2)$ or nothing depending on whether pattern $[f(x)]_0$ first matches $f(2)$ (which makes it possible for the subquery $\nabla[f(1)]_? \cdot h(x)$ to succeed then and return $h(2)$) or $f(1)$ (which causes all executions of subquery $\nabla[f(1)]_? \cdot h(x)$ to fail).

The following theorem states that while evaluation of a deterministic DML query is not itself deterministic, but its results are.

Theorem 6.17. Let Q be a deterministic query in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$. Then $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$ is confluent up to a nominal equivalence. In particular, given ground multisets of facts F , and of fresh facts $F_{\mathbf{n}}$, there is a unique (up to nominal equivalence) term t of the form $\mathbf{n}(G, G_{\mathbf{n}})$ or $\mathbf{f}(F, G_{\mathbf{n}})$ such that $I_Q(F, F_{\mathbf{n}}) \rightarrow^! t$.

Proof:

The only significant difference between the proof of this theorem and Theorem 5.12 is the presence of deletions and fresh facts. The non-confluence introduced by fresh facts can be absorbed with nominal equivalence. Since Q has no deletion conflicts, when a DML subquery $\nabla P \cdot R$ is executed, neither

deletion of facts through P influences execution of R nor execution of R decreases the pool of facts available for matching with P . Moreover, if P contains $[f]_0$ for some fact f , then f is the only fact in P , hence R cannot fail, facts deleted through P are never returned, and $[f]_0$ behaves like $[f]_?$. \square

Let us finish this section with the following remark about expressibility of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$:

Remark 6.18. A typical formalization of database updates is to use pairs of queries which define facts to be, respectively, deleted from, and added to the current database. This approach can be emulated in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$, using multiple DML queries executed in a sequence. First, let us extend the signature Σ with function symbols f^d and f^a for each fact constructor f . Let Q_d and Q_a be queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{qry}}$ which return sets of facts to be deleted and added, respectively, to the database. We assume that Q_d and Q_a contain no subterms of the form $f^d(\vec{t})$ or $f^a(\vec{t})$. Let \hat{Q}_d and \hat{Q}_a be the same as Q_d and Q_a , respectively, except that all subqueries $f(\vec{t})$ of sort Fact are replaced, respectively, with $f^d(\vec{t})$ and $f^a(\vec{t})$. Then to update the database we execute the following DML queries (in this order):

$$\hat{Q}_d, \quad \hat{Q}_a, \quad \nabla[f_1^d(\vec{v}^1)]_? \cdot \nabla[f_1(\vec{v}^1)]_0 \cdot \sqrt{}, \dots, \nabla[f_m^d(\vec{v}^m)]_? \cdot \nabla[f_m(\vec{v}^m)]_0 \cdot \sqrt{}, \\ \nabla[f_1^d(\vec{v}^1)]_0 \cdot \sqrt{}, \dots, \nabla[f_m^d(\vec{v}^m)]_0 \cdot \sqrt{}, \quad \nabla[f_1^a(\vec{v}^1)]_0 \cdot f_1(\vec{v}^1), \dots, \nabla[f_m^a(\vec{v}^m)]_0 \cdot f_m(\vec{v}^m),$$

where f_1, \dots, f_m are fact constructors occurring in Q_d and Q_a . Thus, because of Theorem 5.15 we can express any *relational* database update using multiple DML queries in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$.

7. Reachability analysis of data-centric business processes

In this section we demonstrate the application of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$ in specification and analysis of data-centric business processes. First, we describe a general reachability and simulation framework, and then devote the rest of the section to an extended example specification.

So far, we have specified the rules for execution of a single DML expression. A business process, in general, executes a sequence of DML expressions according to some orchestration rules. A simple example of such rules which we use here, appropriate for a data driven process, is that if a DML expression can be executed successfully then it can be chosen (non-deterministically) as the next command to be executed. Usually (c.f., [40]) such data modifying operations are launched in response to some events, such as user actions which also provide input parameters for the command. In turn, their execution may trigger further events. Here, taking inspiration from [41, 42], we interpret some of the facts as triggering events, user input, and output events (we describe this in more detail later as a part of the example). This simplifies the simulation.

Thus, we specify a business process simply as a finite set Γ of DML expressions in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$. During simulation, at each “business step” a DML expression is chosen non-deterministically and is executed. Unsuccessful execution simply leaves the database of facts unchanged. Alternatively, during reachability analysis which performs a breadth-first search through all possible evolutions of the process it is more efficient to make the system stuck on unsuccessful step. This prunes spurious branches in search tree.

More precisely, a set of DML expressions Γ determines a rewriting system $\mathcal{R}_{\Sigma, \mathcal{D}}(\Gamma)$ defined to be the union of $\mathcal{R}_{\Sigma, \mathcal{D}}^{\text{dml}}(Q)$'s, $Q \in \Gamma$, augmented with two rule schemas

$$\lambda_Q^{\text{new}} : \mathfrak{n}(F, F_{\mathbf{n}}) \Rightarrow I_Q(F, F_{\mathbf{n}}), \quad \lambda_Q^{\text{fail}} : \mathfrak{f}(F, F_{\mathbf{n}}) \Rightarrow I_Q(F, F_{\mathbf{n}}), \quad (31)$$

for all $Q \in \Gamma$.

Rule schema λ_Q^{new} chooses non-deterministically a new DML query for execution and rewrites into an initial state for this query if the execution of the previous one was successful. Similarly, λ_Q^{fail} chooses a new DML query if the execution of the previous one failed. It is important to know that the failure of a randomly chosen DML expression does not usually mean that the business process execution is faulty: Instead, it may simply mean that the given DML expression was not applicable at the moment. Since here the only way to know if the DML expression is applicable is to run it, the rule λ_Q^{fail} is necessary lest the simulation stops prematurely. On the other hand, unsuccessful executions do not change database state, and thus are spurious, adding no useful information. This is why, when doing reachability analysis which explores using breadth-first search all possible paths of execution (in contrast to simulations, where each simulation travels just a single execution path) it is better to drop the rule λ_Q^{fail} .

It is assumed that all DML queries $Q \in \Gamma$ are such that a successful execution of Q consumes and emits a special fact \sharp (of sort `Fact`) called a token. The token does not denote any real data, but rather facilitates a non-deterministic choice of user input. Say, if in the database there were facts $f(a_1), f(a_2), \dots$, where a_1, a_2, \dots are possible user inputs for some business step, then we can simulate user choice and execution of further action $D(x)$ (based on this choice and expressed as DML query with free variable x storing user's decision) by using the DML expression of the form $\nabla[\sharp]_0 \circ [f(x)]_? \cdot D(x)$. If the query wouldn't match and remove the token then the action $D(x)$ would be executed for every possible user input. In the example described in this section instead of a constant token we use tokens $\sharp : \text{Nat} \rightarrow \text{Fact}$ parametrized by a natural number. All DML queries consume a token with a non-zero parameter and emit a token with a parameter decreased by one. This permits limiting the number of "large business steps", i.e., executions of DML queries, in the simulation or search procedure. Rewriting systems such as Maude permit limiting rewriting steps in the search procedure. However, execution of each DML query can take many rewriting steps, the number of which is not easy to estimate. Thus, it is not trivial to pass from the number of rewriting steps to the number of business steps (which are more natural in this context).

Given an initial database F we start reachability analysis from term $\mathfrak{n}(F \circ \sharp(k), C_{s_1}(v_{m_1}^{s_1}) \circ \dots \circ C_{s_n}(v_{m_n}^{s_n}))$, where k is the maximal depth of search expressed in the number of business steps, s_1, \dots, s_n are nominal sorts for which we need fresh values, and m_1, \dots, m_n are such that values $v_{p_i}^{s_i}$ for $p_i \geq m_i, i \in \{1, \dots, n\}$, do not occur in F . In case of reachability analysis we search for the term of the form $\mathfrak{n}(F, F_{\mathbf{n}})$ where the database F satisfies some condition (either desirable or undesirable one).

7.1. Example specification

We borrow an example from [28, Appendix C] to demonstrate specification of a business process as a set of DML expressions in $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{dml}}$. The example concerns the process of selecting and advertising

restaurant offers of dinners by employees of mediating agency, and managing corresponding bookings. The lifecycles of two key artifact types — *Offer* and *Booking* — are presented as finite state machines in Figure 1. Each agent publishes exactly one restaurant offer — either the new one which just came or the one which was previously put on hold. The published offer is in the state *available*. Agent puts the offer he currently publishes on hold (state *onHold*) when picking up another offer for publication. Dashed arrows in Figure 1 indicate that entering a given state by an artifact may trigger state change in another artifact, e.g., there is a dashed arrow between the *available* state and the anonymous transition into *onHold* state (in a distinct artifact of type *Offer*). *Available* offer may get closed (state *closed*, or be picked up by a customer (transition *newBooking* to state *beingBooked*). The latter triggers creation of a new *Booking* artifact. *Booking* starts with a preliminary phase of *drafting* (state *drafting*) in which the customer chooses dinner hosts (transition *addHosts*). After draft submission (which changes the state to *submitted*) the agent computes price for the offer (transition *determineProposal* to state *finalized*) and the customer decides to either accept or reject the proposal transitioning, respectively, to the *accepted* or *anceled* state. The acceptance may in some cases go through *toBeValidated* state when additional validation is necessary.

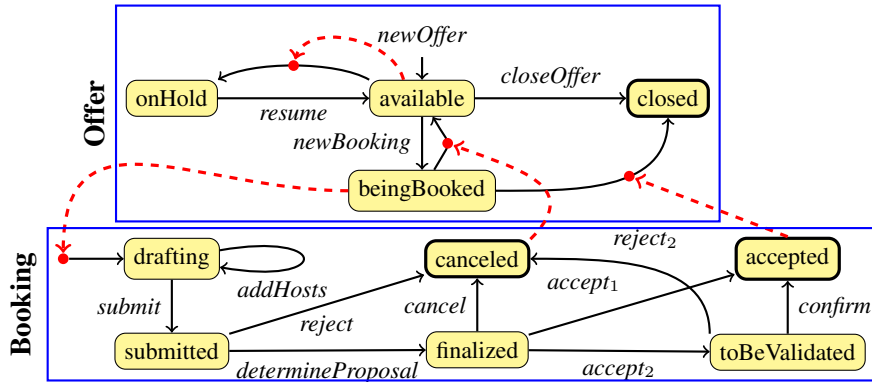


Figure 1. Lifecycles of *Offer* and *Booking* artifacts presented as finite state machines [28, Figure 5] (see also [43, Figure 5])

States of offers and bookings are constants of sorts *OState* and *BState*, respectively, named like in Figure 1. We use the following nominal sorts for identifiers: *Rest* for restaurants; *Person* for customers, agents and hosts; *Offer*, *Book* and *Url* for offers, bookings, and url's of finalized proposals, respectively. For facts we use the following constructors:

$$\begin{aligned}
 & \text{Rest} : \text{Rest} \rightarrow \text{Fact}, \quad \text{Agent, Cust} : \text{Person} \rightarrow \text{Fact}, \\
 & \text{Offer} : \text{Offer } \text{OState } \text{Rest } \text{Person} \rightarrow \text{Fact}, \quad \text{Book} : \text{Book } \text{BState } \text{Offer } \text{Person} \rightarrow \text{Fact}, \\
 & \text{Host} : \text{Book } \text{Person} \rightarrow \text{Fact}, \quad \text{Prop} : \text{Book } \text{Url} \rightarrow \text{Fact}.
 \end{aligned}$$

where facts $\text{Rest}(r)$, $\text{Agent}(a)$, $\text{Cust}(c)$ indicate that r , a , and c are, respectively, identifiers of a registered restaurant, agent, and customer. $\text{Offer}(o, s, r, a)$ means that an offer o in a state s for a restaurant r is managed by an agent a . $\text{Book}(b, s, o, c)$ means that booking b in a state s for a customer

c is related to an offer o . A fact $\text{Host}(b, p)$ indicates that a person p is included as a host for booking b . Finally, $\text{Prop}(b, u)$ indicates that finalized proposal for booking b , with details and prices, is available at the url u .

We now specify selected transitions from Figure 1 in detail. Transition `newOffer` responsible for creation of new offers is implemented with the following DML query:

$$\begin{aligned} & \nabla[\sharp(s(n))]_0 \circ [C_{\text{Offer}}(o)]_n \circ [\text{Agent}(a)]_? \circ [\text{Rest}(r)]_! \cdot (\forall [\text{Offer}(o', \text{beingBooked}, r', a)]_? \cdot \perp) \\ & \Rightarrow (O(o, \text{available}, r, a) \triangleright (\nabla[O(o', \text{available}, r', a)]_0 \cdot O(o', \text{onHold}, r', a)) \triangleright \sharp(n)). \end{aligned}$$

Above, $s : \text{Nat} \rightarrow \text{Nat}$ denotes the successor function. We assume that each DML query is executed against a database in which there is exactly one token matching $\sharp(s(n))$, i.e., a token holding a number greater than zero. Thus, in the above, we first choose a *single* fresh offer identifier, and, non-deterministically, a *single* registered agent and a *single* restaurant. The token is marked with $[-]_0$, so it is removed from the database after matching (this guarantees that we choose no more than one agent, restaurant, and offer identifier). The query emits back a token with a number decreased by 1 ensuring the possibility (if this number is greater than zero) of executing a next query. The restaurant can be arbitrary (as long as it is registered in the system), however the agent must not manage an offer being booked, as described by the deterministic condition

$$\forall[O(o', \text{beingBooked}, r', a)]_? \cdot \perp$$

inside the above DML query. If this condition is not satisfied, the quantifier step fails, the token is returned to the database and a new matching is tried. Since $\text{Agent}(a)$ is marked by $[-]_?$, we do not try the same agent again. If the correct matching is found, a fact describing new offer is added to the database. We also change the state of any available offer managed by the agent of the new offer to `onHold`.

An offer which was put on hold, may be resumed by any agent who is not managing an offer which is currently being booked. Agent resuming an offer becomes the new manager of the offer:

$$\begin{aligned} & \nabla[\sharp(s(n)) \circ \text{Offer}(o, \text{onHold}, r, a)]_0 \circ [\text{Agent}(a')]_? \cdot (\forall [\text{Offer}(o', \text{beingBooked}, r', a')]_? \cdot \perp) \\ & \Rightarrow (\sharp(n) \triangleright \text{Offer}(o, \text{available}, r, a') \\ & \triangleright (\nabla[\text{Offer}(o', \text{available}, r', a')]_0 \cdot \text{Offer}(o', \text{onHold}, r', a'))). \end{aligned}$$

With the `newBooking` transition some offer o changes state from `available` to `beingBooked`. It also triggers creation of a new booking (with a fresh identifier) in the `drafting` state for the chosen offer o on behalf of some registered customer:

$$\begin{aligned} & \nabla[\sharp(s(n)) \circ \text{Offer}(o, \text{available}, r, a)]_0 \circ [C_{\text{Book}}(b)]_n \circ [\text{Cust}(c)]_! \\ & \cdot (\text{Offer}(o, \text{beingBooked}, r, a) \triangleright \text{Book}(b, \text{drafting}, o, c) \triangleright \sharp(n)). \end{aligned}$$

The customer involved in booking can add dinner hosts one by one (see transition `addHosts` in Figure 1) as long as the booking is in the `drafting` stage. The added host can be either fresh or be

present in the database as a host for another offer. We use separate DML queries for each of those cases. The first case (of a fresh host) is trivial:

$$\nabla[\sharp(s(n))]_0 \circ [C_{\text{Person}}(h)]_n \circ [\text{Book}(b, \text{drafting}, o, c)]_! \cdot (\sharp(n) \triangleright \text{Host}(b, h)).$$

In the second case we have to ensure that we are not adding the same person twice:

$$\nabla[\sharp(s(n))]_0 \circ [\text{Book}(b, \text{drafting}, o, c)]_! \circ [\text{Host}(b', h)]_?. ((\forall [\text{Host}(b, h)]_?. \perp) \Rightarrow (\sharp(n) \triangleright \text{Host}(b, h))).$$

The submit action simply changes the state of the booking from `drafting` to `submitted`. Then, if the customer's customized booking is infeasible, it can be rejected (`reject` transition in Figure 1, the implementation of which we omit for brevity's sake). Otherwise, the final proposal (which includes cost, etc.) to the customer who owns the booking is created (transition `determineProposal` in Figure 1). The preparation of the proposal is abstracted as (1) creating the fresh url to the proposal, and (2) removing information about hosts (which is available at the new url). As before, we pick the booking non-deterministically using the token and appropriate pattern:

$$\begin{aligned} \nabla[\sharp(s(n)) \circ \text{Book}(b, \text{submitted}, o, c)]_0 \circ [C_{\text{Url}}(u)]_n. \\ (\text{Prop}(b, u) \triangleright \text{Book}(b, \text{finalized}, o, c) \triangleright (\nabla[\text{Host}(b, h)]_0 \cdot \surd) \triangleright \sharp(n)). \end{aligned}$$

A finalized booking proposal for a restaurant r can be accepted either immediately (with `accept1`) or after an additional confirmation (with `accept2`). The first case applies only to golden customers of a given restaurant r , i.e., those who successfully booked a dinner in r at least k -times, for some fixed k . Accepting a proposal changes the state of the offer to which the booking belongs to `closed`:

$$\begin{aligned} \nabla[\sharp(s(n)) \circ \text{Book}(b, \text{finalized}, o, c) \circ \text{Offer}(o, \text{beingBooked}, r, a)]_0 \circ [\text{Cust}(c)]_?. \\ (\exists [\text{Offer}(o_1, \text{closed}, r, a_1) \circ B(b_1, \text{accepted}, o_1, c) \\ \circ \dots \circ \text{Offer}(o_k, \text{closed}, r, a_k) \circ \text{Book}(b_k, \text{accepted}, o_k, c)]_?. \top) \\ \Rightarrow (\text{Book}(b, \text{accepted}, o, c) \triangleright \text{Offer}(o, \text{closed}, r, a) \triangleright \sharp(n)). \end{aligned}$$

Remark 7.1. In our earlier work [15] we have used the almost same example (with minor modifications) to illustrate an alternative formalism (c.f. Section 1.1 in the current paper) where queries, also implemented in the rewriting system, but using meta-level features, are deterministic. This makes them behave like a classical queries, but because of determinism it is not possible to simulate user input directly. Instead, a separate mechanism had to be introduced to simulate non-deterministic input choice. Secondly, DML expressions in [15] did not add or, more importantly, delete facts from the database directly. Instead, they return pairs (which need to be specified in the DML expression itself) of (multi)sets of facts: those to be deleted and those to be added. However, in this particular example (and we believe it is typical) facts to be deleted are matched by parts of the patterns in the query. This (in [15]) led to code duplication, and suggested natural use of rewriting rules (which, of course, replaces matched subterms), and ultimately led to the formalism described in this paper.

Remark 7.2. We have implemented both the syntax and semantics of $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{Cnd}}$ and $\mathcal{Q}_{\Sigma, \mathcal{D}}^{\text{Dml}}$ in Maude [44]. The implementation is available on the project's website [45]. To test the implementation we have

used specification of the business process described above (also available from [45]). The specification compiles into a Maude's system module which contains definitions of 196 operators, 285 equations and 83 rewriting rules (in actual implementation we used equations in place of deterministic rewriting rules).

Let us now describe a simple example of a reachability analysis with the specification just described. Let

$$\text{initDB} := \text{agent}(a_1) \circ \text{agent}(a_2) \circ \text{cust}(c_1) \circ \text{cust}(c_2) \circ \text{rest}(r_1) \circ \text{rest}(r_2) \circ \sharp(7)$$

be an initial database of facts. Let

$$\text{initDBN} := C(i_0^{\text{Offer}}) \circ C(i_0^{\text{Book}}) \circ C(i_0^{\text{Url}}) \circ C(i_0^{\text{Person}})$$

be an initial multiset of fresh facts. Finally, let

$$\text{initState} := \mathbf{n}(\text{initDB}, \text{initDBN})$$

be an initial state. Note that since the token is parametrized by 7, it follows that we can make at most seven successful business steps from this state. We are interested in checking if we can reach from initState (in no more than 7 business steps) the state in which there exists a closed offer (and accepted associated booking) from the database with no bookings and offers. Formally, we want to reach the state matching

$$\mathbf{n}(F \circ \text{Offer}(o_1, \text{closed}, r_1, a_1) \circ \text{Book}(b_1, \text{accepted}, o_1, c_1), F_{\mathbf{n}}).$$

Using our implementation [45] we can easily check that a matching state is indeed reachable in 6 business steps (Maude reported 525165 actual rewritings in 2528ms).

8. Conclusion

We have presented a multiset non-deterministic query and data manipulation language $\mathcal{Q}_{\Sigma, \mathcal{D}}$ based on conditional term rewriting. The intended application of this language is in specification, simulation and reachability analysis of data-centric business processes. However, the remarkable features of $\mathcal{Q}_{\Sigma, \mathcal{D}}$, particularly non-determinism and non-standard approach to variable binding, make it interesting on its own. We show that non-determinism of queries is useful for simulating user choices, but we also provide easily identifiable syntactic restrictions which ensure uniqueness of query results. Interestingly, this non-determinism leads to bisimulation-like definitions of logical equivalence between formulas. In the last section we demonstrated how sets of DML queries can be used to specify a business process and we provide a simple framework for simulation and testing.

$\mathcal{Q}_{\Sigma, \mathcal{D}}$ is a multiset query language. Most formal query languages, including relational calculus and algebra, are based on sets. One under-appreciated fact is that SQL is really a multiset query language, and for a very good reason — removing duplicates is expensive. While this was not our primary reason to use multisets, we believe that using multiset languages encourages query design

which avoids unnecessary expensive operations, and takes the complexity of query execution into account better than set-based languages.

The fact that closed $Q_{\Sigma, \mathcal{D}}$ formulas are compiled to rewriting systems permits their symbolic execution using narrowing [16]. We intend to explore this possibility in future research. This is also one of the reasons why it was important to limit the use of conditional rules as much as possible: many implementations of narrowing (see e.g., [46]) do not permit narrowing with conditions.

We have implemented $Q_{\Sigma, \mathcal{D}}^{\text{cnd}}$, $Q_{\Sigma, \mathcal{D}}^{\text{dml}}$ and a specification framework extending the one described at the beginning of Section 7 in Maude [46]. The implementation is available from [45]. It differs in non-essential way from the one described in the present paper, but the code is extensively documented.

Acknowledgements

The author is grateful to the anonymous reviewers for their helpful remarks

References

- [1] Hull R. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman R, Tari Z (eds.), *On the Move to Meaningful Internet Systems: OTM 2008*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008 pp. 1152–1163. doi:10.1007/978-3-540-88873-4_17.
- [2] Calvanese D, De Giacomo G, Montali M. Foundations of Data-aware Process Analysis: A Database Theory Perspective. In: *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '13*. ACM, New York, NY, USA, 2013 pp. 1–12. doi:10.1145/2463664.2467796.
- [3] van der Aalst WM. The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, 1998. **8**(01):21–66. doi:10.1142/S0218126698000043.
- [4] van der Aalst WM, Ter Hofstede AH. YAWL: yet another workflow language. *Information systems*, 2005. **30**(4):245–275. doi:10.1016/j.is.2004.02.002.
- [5] Rosa-Velardo F, de Frutos-Escrig D. Decidability and complexity of Petri nets with unordered data. *Theoretical Computer Science*, 2011. **412**(34):4439–4451.
- [6] Lasota S. Decidability border for Petri nets with data: WQO dichotomy conjecture. In: *International Conference on Application and Theory of Petri Nets and Concurrency*. Springer, 2016 pp. 20–36. doi:10.1007/978-3-319-39086-4_3.
- [7] Montali M, Rivkin A. Model checking Petri nets with names using data-centric dynamic systems. *Formal Aspects of Computing*, 2016. **28**(4):615–641. doi:10.1007/s00165-016-0370-6.
- [8] Montali M, Rivkin A. DB-Nets: On the Marriage of Colored Petri Nets and Relational Databases. In: *Transactions on Petri Nets and Other Models of Concurrency XII*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017 pp. 91–118. doi:10.1007/978-3-662-55862-1_5.
- [9] Montali M, Rivkin A. From DB-nets to Coloured Petri Nets with Priorities. In: *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2019 pp. 449–469. doi:10.1007/978-3-030-21571-2_24.
- [10] Meseguer J. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 1992. **96**(1):73–155. doi:10.1016/0304-3975(92)90182-F.

- [11] Meseguer J, Rosu G. The rewriting logic semantics project. *Theoretical Computer Science*, 2007. **373**(3):213 – 237. doi:10.1016/j.tcs.2006.12.018.
- [12] Stehr MO, Meseguer J, Ölveczky PC. Rewriting Logic as a Unifying Framework for Petri Nets. In: *Unifying Petri Nets: Advances in Petri Nets*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001 pp. 250–303. doi:10.1007/3-540-45541-8_9.
- [13] Padberg J, Schulz A. Model Checking Reconfigurable Petri Nets with Maude. In: Echahed R, Minas M (eds.), *Graph Transformation*. Springer International Publishing, Cham, 2016 pp. 54–70. doi:10.1007/978-3-319-40530-8_4.
- [14] Kheldoun A, Barkaoui K, Ioualalen M. Formal verification of complex business processes based on high-level Petri nets. *Information Sciences*, 2017. **385**:39–54. doi:10.1016/j.ins.2016.12.044.
- [15] Zieliński B. A Query Language Based on Term Matching and Rewriting. *Fundamenta Informaticae*, 2019. **169**:237–274. doi:10.3233/FI-2019-1845.
- [16] Meseguer J, Thati P. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 2007. **20**(1-2):123–160. doi:10.1007/s10990-007-9000-6.
- [17] Fay M. First-order unification in an equational theory. In: *Proceedings of the 4th Workshop on Automated Deduction*, Austin, Texas, 1979.
- [18] Hullot JM. Canonical forms and unification. In: *International Conference on Automated Deduction*. Springer, 1980 pp. 318–334. doi:10.1007/3-540-10009-1_25.
- [19] Alpuente M, Escobar S, Iborra J. Termination of narrowing revisited. *Theoretical Computer Science*, 2009. **410**(46):4608–4625. doi:10.1016/j.tcs.2009.07.037.
- [20] Zieliński B, Maślanka P. Relational Transition System in Maude. In: *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation: 13th International Conference, BDAS 2017, Ustroń, Poland, May 30 - June 2, 2017, Proceedings*. Springer International Publishing, Cham, 2017 pp. 497–511. doi:10.1007/978-3-319-58274-0_39.
- [21] Roşu G, Ellison C, Schulte W. Matching Logic: An Alternative to Hoare/Floyd Logic. In: *Algebraic Methodology and Software Technology*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011 pp. 142–162. doi:10.1007/978-3-642-17796-5_9.
- [22] Roşu G. Matching logic. *arXiv:1705.06312*, 2017.
- [23] Stehr MO. CINNI-A Generic Calculus of Explicit Substitutions and its Application to λ - ζ -and π -Calculi. *Electronic Notes in Theoretical Computer Science*, 2000. **36**:70–92. doi:10.1016/S1571-0661(05)80125-2.
- [24] Baader F. *The description logic handbook: Theory, implementation and applications*. Cambridge University Press, New York, NY, USA, 2003. ISBN:0-521-78176-0.
- [25] De Giacomo G, De Masellis R, Rosati R. Verification of conjunctive artifact-centric services. *International Journal of Cooperative Information Systems*, 2012. **21**(02):111–139. doi:10.1142/S0218843012500025.
- [26] Hariri BB, Calvanese D, De Giacomo G, De Masellis R, Felli P. Foundations of relational artifacts verification. In: *International Conference on Business Process Management*. Springer, 2011 pp. 379–395. doi:10.1007/978-3-642-23059-2_28.

- [27] Calvanese D, Montali M, Patrizi F, De Giacomo G. Description logic based dynamic systems: modeling, verification, and synthesis. In: Proceedings of the 24th International Conference on Artificial Intelligence. AAAI Press, 2015 pp. 4247–4253.
- [28] Abdulla PA, Aiswarya C, Atig MF, Montali M, Rezine O. Recency-Bounded Verification of Dynamic Database-Driven Systems. In: Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '16. ACM, New York, NY, USA, 2016 pp. 195–210. doi: 10.1145/2902251.2902300.
- [29] Chen-Burger YH, Robertson D. Automating business modelling: a guide to using logic to represent informal methods and support reasoning. Springer Science & Business Media, 2006. doi:10.1007/b138799.
- [30] Merouani H, Mokhati F, Seridi-Bouchelaghem H. Formalizing Artifact-Centric Business Processes - Towards a Conformance Testing Approach. In: Proceedings of the 16th International Conference on Enterprise Information Systems. 2014 pp. 368–374. doi:10.5220/0004951803680374.
- [31] McCarthy J, Hayes PJ. Some philosophical problems from the standpoint of artificial intelligence. *Readings in artificial intelligence*, 1969. pp. 431–450.
- [32] Deutsch A, Li Y, Vianu V. Verifas: a practical verifier for artifact systems. *Proceedings of the VLDB Endowment*, 2017. **11**(3):283–296. doi:10.14778/3157794.3157798.
- [33] Deutsch A, Li Y, Vianu V. Verification of hierarchical artifact systems. *ACM Transactions on Database Systems (TODS)*, 2019. **44**(3):1–68. doi:10.1145/3321487.
- [34] Calvanese D, Ghilardi S, Gianola A, Montali M, Rivkin A. Formal modeling and SMT-based parameterized verification of data-aware BPMN. In: International Conference on Business Process Management. Springer, 2019 pp. 157–175. doi:10.1007/978-3-030-26619-6_12.
- [35] Seco JC, Debois S, Hildebrandt T, Slaats T. RESEDA: Declaring live event-driven computations as REactive SEmi-structured DATA. In: 2018 IEEE 22nd International enterprise distributed object computing conference (EDOC). IEEE, 2018 pp. 75–84. doi:10.1109/EDOC.2018.00020.
- [36] Meseguer J. Membership algebra as a logical framework for equational specification. In: Recent Trends in Algebraic Development Techniques. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998 pp. 18–61. doi:10.1007/3-540-64299-4_26.
- [37] Huet G. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *Journal of the ACM (JACM)*, 1980. **27**(4):797–821.
- [38] Thielscher M. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 1998. **2**(3-4):179–192.
- [39] Ochremiak J. Nominal sets over algebraic atoms. In: International Conference on Relational and Algebraic Methods in Computer Science. Springer, 2014 pp. 429–445. doi:10.1007/978-3-319-06251-8_26.
- [40] Hull R, Damaggio E, De Masellis R, Fournier F, Gupta M, Heath III FT, Hobson S, Linehan M, Maradugu S, Nigam A, et al. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: Proceedings of the 5th ACM international conference on Distributed event-based system. ACM, 2011 pp. 51–62. doi:10.1145/2002259.2002270.
- [41] Abiteboul S, Vianu V, Fordham B, Yesha Y. Relational Transducers for Electronic Commerce. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '98. ACM, New York, NY, USA. ISBN 0-89791-996-3, 1998 pp. 179–187. doi: 10.1145/275487.275507.

- [42] Abiteboul S, Vianu V, Fordham B, Yesha Y. Relational transducers for electronic commerce. *Journal of Computer and System Sciences*, 2000. **61**(2):236–269. doi:10.1006/jcss.2000.1708.
- [43] Abdulla PA, Aiswarya C, Atig MF, Montali M, Rezine O. Recency-bounded verification of dynamic database-driven systems (extended version). *arXiv preprint arXiv:1604.03413*, 2016.
- [44] Clavel M, Durán F, Eker S, Lincoln P, Martí-Oliet N, Meseguer J, Talcott C. The Maude 2.0 System. In: Nieuwenhuis R (ed.), *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in *Lecture Notes in Computer Science*. Springer-Verlag, 2003 pp. 76–87. doi:10.1007/3-540-44881-0_7.
- [45] Zieliński B. Nondeterministic Rewriting Query Language (NDRQL). Project website, <http://ki.wfi.uni.lodz.pl/ndrql/>.
- [46] Clavel M, Durán F, Eker S, Escobar S, Lincoln P, Martí-Oliet N, Meseguer J, Talcott C. *Maude Manual (Version 2.7.1)*, 2016.