Semantic Reasoning with SPARQL in Heterogeneous Multi-Context Systems[*]

Peter Schüller and Antonius Weinzierl

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{schueller,weinzierl}@kr.tuwien.ac.at

Abstract. Multi-Context Systems (MCSs) are an expressive framework for interlinking heterogeneous knowledge systems, called contexts. Possible contexts are ontologies, relational databases, logic programs, RDF triplestores, etc. MCSs contain bridge rules to specify knowledge exchange between contexts. We extend the MCS formalism and propose SPARQL-MCS where knowledge exchange is specified in the style of SPARQL CONSTRUCT queries. Different from previous approaches to variables in MCSs, we do not impose any restrictions on contexts. To achieve this, we introduce a general approach for variable substitutions in heterogeneous systems. We define syntax and semantics of SPARQL-MCS and investigate fixpoint evaluation of monotonic MCSs.

1   Introduction

Multi-Context Systems (MCSs) [4] are the result of a successful line of research [9, 12, 5] to describe interlinking of heterogeneous knowledge bases (called contexts) using (possibly nonmonotonic) bridge rules.

Intuitively, a bridge rule $1 : u \leftarrow (2 : v), not(3 : w)$ at context C_1 adds *formula* u to the knowledge base of C_1 whenever *belief* v is *accepted* by context C_2 and w is *not accepted* by C_3. However, MCS bridge rules are not designed to contain variables, which limits the applicability of the formalism in practice.

In this paper we extend MCS and introduce SPARQL-MCS, where information flow is specified using SPARQL queries with variables. We propose SPARQL-MCS, where knowledge exchange is specified by SPARQL bridges of the form

CONSTRUCT $2 :\{$ *?Person* $\langle \#needsMed \rangle$ *?Med.* $\}$
      WHERE $\{$ $(1 : person(?Person, ?Id))$ AND $(3 : require(?Id, ?Med))$ $\}$

which intuitively states that an RDF-triple[1] *?Person* $\langle \#needsMed \rangle$ *?Med* is added to context C_2 if context C_1 believes an atom *person* and context C_3

---

[1] RDF is a semantic web framework for storing information as graphs, encoded in (subject predicate object) triples.

believes an atom *require* such that their join (over contexts) yields a mapping of variables ?*Person* and ?*Med* that can be substituted into the RDF-triple.

While this example looks quite simple, it already demonstrates a difficulty arising from the combination of variables and heterogeneous contexts, namely that values must be converted between formalisms. For example a person *sue* must be represented as an IRI in an RDF-triple, while a database might require a format different from $\langle \#sue \rangle$. Furthermore, substituting variables into formulas and beliefs requires knowledge about the inner structure of these objects: replacing variable ?X in formula $p("foo?X", ?X)$ is not a matter of simple search-and-replace, depending on the application scenario, one or both occurances of ?X might be a target for variable substitution.

Both issues are likely to arise in the heterogeneous setting of MCSs where different formalisms are interlinked. These issues are not addressed in a satisfactory way by existing proposals to add variables to MCSs: in [7], variables are seen as schematic variables only, i.e., a bridge rule is only a short notation for the set of instantiated (thus variable-free) bridge rules. In [8], bridge rules with variables require that the contexts of the MCS adhere to a given syntax for predicates; and similar in [13] where only first order languages are considered. The first solution is infeasible for large numbers of instantiations, while the second and third cannot capture formalisms that do not use predicate syntax, e.g., RDF stores.

In this work, we propose a uniform solution for combining heterogeneous logics with variables, thereby solving both issues, and apply that solution in the SPARQL-MCS formalism. Our main contributions are as follows.

- We introduce the syntax of SPARQL-MCS where information flow is specified using SPARQL queries. To that end we extend the core fragment of SPARQL (cf. [2]) to query arbitrary data sources, without restricting to first-order theories.
- To handle variables and variable substitutions in heterogeneous systems, we propose a framework for variable substitutions. The framework uses a global set of (abstract) entities that can be substituted for variables, and consists of two functions for
  - applying variable substitutions to expressions, and for
  - matching an expression with beliefs of a context to obtain variable substitutions.

  This framework allows us to view expressions with variables as objects without inner structure, therefore we are able to introduce variables without adding restrictions on contexts. Furthermore, our approach captures conversion between entities in different logics.
- We give a declarative characterization of the semantics of SPARQL-MCS.
- For the special case of monotonic SPARQL-MCS, we give a fixed-point characterization of its semantics.

2   Preliminaries

A heterogeneous nonmonotonic MCS [4] consists of *contexts*, each composed of a knowledge base with an underlying *logic*, and a set of *bridge rules* which control

the information flow between contexts. A logic $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$ is an abstraction, which allows to capture many monotonic and nonmonotonic logics, e.g., classical logic, description logics, default logics. It consists of the following components; the first two intuitively describe syntax, the third semantics:

- $\mathbf{KB}_L$ is the set of well-formed knowledge bases of L. We assume each element of $\mathbf{KB}_L$ is a set of "formulas".
- $\mathbf{BS}_L$ is the set of possible belief sets, where a belief set is a set of "beliefs".
- $\mathbf{ACC}_L : \mathbf{KB}_L \rightarrow 2^{\mathbf{BS}_L}$ is a function describing the semantics of the logic by assigning to each knowledge base a set of acceptable belief sets.

Each context has its own logic, which allows to model heterogeneous systems.

Example 1. All kinds of logics can be captured with this definition. In the following we present specific logics, which we use in later examples in this paper.

i) For quantifier- and function-free, predicate logic over language Σ, the logic $L_{CL} = (\mathbf{KB}_{CL}, \mathbf{BS}_{CL}, \mathbf{ACC}_{CL})$ is such that $\mathbf{KB}_{CL}$ is the set of well-formed Σ-formulas (using predicates, and variables, but no function symbols or quantifiers), $\mathbf{BS}_{CL}$ is the set of sets of atoms wrt. Σ, and for each $kb \in \mathbf{KB}_{CL}$ is $N \in \mathbf{ACC}_{CL}(kb)$ iff N is a model of kb under the closed-world assumption.

ii) For an RDF triplestore [2], the logic $L_{RDF} = (\mathbf{KB}_{RDF}, \mathbf{BS}_{RDF}, \mathbf{ACC}_{RDF})$ is such that $\mathbf{KB}_{RDF}$ consists of all valid RDF-triplestore contents, including syntax like '$\langle \#foo \rangle \ \langle \#bar \rangle \ 1 \ ; \ \langle \#baz \rangle$ 'abc' .'. $\mathbf{BS}_{RDF}$ contains all valid RDF graphs, i.e., sets of RDF triples (without shortcut syntax). $\mathbf{ACC}_{RDF}$ accepts for each element in $\mathbf{KB}_{RDF}$ the corresponding RDF graph from $\mathbf{BS}_{RDF}$.

iii) For an $\mathcal{AL}$ description logic [3] the logic $L_{\mathcal{AL}}$ is such that $\mathbf{KB}_{\mathcal{AL}}$ is the set of well-formed theories in $\mathcal{AL}$, $\mathbf{BS}_{\mathcal{AL}}$ is the powerset of the set of well-formed assertions $C(o)$ with C a concept name and o an individual name of $\mathcal{AL}$, and $\mathbf{ACC}_{\mathcal{AL}}(kb)$ returns the set of concept assertions entailed by kb. $\square$

Ordinary MCSs (in the sense of [4]) model information flow between contexts using *bridge rules* without variables: a bridge rule can add information to a context, depending on the belief sets accepted at other contexts [4]. Let $\mathcal{L} = (L_1, \ldots, L_n)$ be a tuple of logics. An L_k-*bridge rule* r over $\mathcal{L}$ is of the form $(k:s) \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \mathbf{not} \ (c_{j+1} : p_{j+1}), \ldots, \mathbf{not} \ (c_m : p_m)$ where $1 \leq c_i \leq n$, p_i is an element of some belief set of L_{c_i}, and k refers to the context receiving formula s. We denote by $h_b(r)$ the formula s in the head of r.

An *MCS* [4] $M = (C_1, \ldots, C_n)$ is a collection of contexts $C_i = (L_i, kb_i, br_i)$, $1 \leq i \leq n$, where $L_i = (\mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i)$ is a logic, $kb_i \in \mathbf{KB}_i$ a knowledge base, and br_i is a set of L_i-bridge rules over $(L_1, \ldots, L_n)$.

Example 2. Let $M = (C_1, C_2)$ be an MCS where C_1 is an RDF triplestore about persons and their social security numbers while C_2 is a propositional logic checking the completeness of C_1; the respective logics are as in Example 1; knowledge bases are $kb_1 = \{\langle \#reg \rangle \ \langle \#person \rangle \ sue. \ \langle \#sue \rangle \ \langle \#ssn \rangle \ 123.\}$ and $kb_2 = \{sue \wedge has\_ssn \rightarrow data\_complete\}$. The information flow is given by two bridge rules of C_2 which are $(2 : sue) \leftarrow (1 : \langle \#reg \rangle \ \langle \#person \rangle \ sue.)$ and $(2 : has\_ssn) \leftarrow (1 : \langle \#sue \rangle \ \langle \#ssn \rangle \ 123.)$ $\square$

Let $M = (C_1, \ldots, C_n)$ be an MCS, a sequence $S = (S_1, \ldots, S_n)$ with $S_i \in \mathbf{BS}_i$ is called a *belief state* of M. Semantics of MCSs is defined in terms of *equilibria*, intuitively, a belief state S which is acceptable at every context given the information exchange of applicable bridge rules. A bridge rule r of context C_k is *applicable* in S if its positive (negative) body literals are present (absent) at their respective context's belief states, i.e., for a L_k bridge rule (over $\mathcal{L}$ as before) we write $r \in app(br_i, S)$ iff $p_\ell \in S_{c_\ell}$ for $1 \leq \ell \leq j$ and $p_\ell \notin S_{c_\ell}$ for $j < \ell \leq n$.

A belief state $S = (S_1, \ldots, S_n)$ of M is an *equilibrium* [4] iff, for $1 \leq i \leq n$, the following condition holds: $S_i \in \mathbf{ACC}_i(kb_i \cup \{hd(r) \mid r \in app(br_i, S)\})$.

Example 3. In Example 2 there is one equilibrium $S = (S_1, S_2)$ with $S_1 = kb_1$ and $S_2 = \{sue, has\_ssn, data\_complete\}$. Both bridge rules are applicable in S. $\quad\square$

3 SPARQL-based Multi-Context Systems

SPARQL-MCS use the same formalism as ordinary MCSs for abstracting from contexts, but they use SPARQL queries to specify information flow and they allow the use of variables without imposing any restrictions on contexts. Note that, for simplicity, we do not consider blank nodes or value invention in our SPARQL-MCS. We proceed with a concrete example of a SPARQL-MCS and later on present all technical details for variable substitution, syntax and semantics of SPARQL-MCS.

Example 4. Our running example is a hospital information system M consisting of the following contexts: a patient data RDF triplestores $C_{patients}$, a laboratory test triplestore C_{lab}, a disease description logic ontology C_{onto}, and a (classical logic) decision support system C_{dss} which suggests proper treatments for patients. For brevity, we only consider patient 'Sue'. The knowledge bases are as follows:

$$
\begin{aligned}
kb_{patients} &= \{\langle\#sue\rangle \ \langle\#ssn\rangle \ 123 \ ; \ \langle\#allergy\rangle \ ab\_s \ . \ \}, \\
kb_{lab} &= \{\langle\#sue\rangle \ \langle\#xray\rangle \ pneumonia \ . \ \}, \\
kb_{onto} &= \{Pneumonia \sqcap Marker1 \sqsubseteq AtypPneumonia\}, \\
kb_{dss} &= \{need(Id, ab) \rightarrow (give(Id, ab\_s) \vee give(Id, ab2)), \\
&\qquad \neg(\ give(Id, ab\_s) \wedge give(Id, ab2) \ ), \\
&\qquad \neg(\ forbid(Id, Drug) \wedge give(Id, Drug) \ ) \ \}.
\end{aligned}
$$

$C_{patients}$ and C_{lab} are RDF triplestores using L_{RDF} (see Ex. 1). $C_{patients}$ stores Sue's social security number and an allergy to antibiotic 'ab_s'. C_{lab} stores blood and X-ray examinations results: '$pneumonia$' was detected in Sue's X-ray. C_{onto} uses $L_{\mathcal{AL}}$ and specifies that presence of a blood marker in combination with pneumonia indicates atypical pneumonia. C_{dss} uses L_{CL} and suggests medications using *give*, e.g., adding $\{need(sue, ab)\}$ to kb_{dss} causes C_{dss} to accept belief sets $\{need(sue, ab), give(sue, ab\_s)\}$ and $\{need(sue, ab), give(sue, ab2)\}$.

We have the following knowledge interlinking between contexts: C_{onto} imports X-ray and blood test results from $C_{patients}$, we indicate a need for antibiotic medication in C_{dss} whenever C_{onto} classifies a patient as having pneumonia, and

we specifically require ab_s for atypical pneumonia. Furthermore, in C_{dss} we forbid administration of all drugs where an allergy is stored in $C_{patients}$.

Knowledge exchange is formulated with SPARQL-bridges as follows:

$$
\begin{aligned}
r_1 = \;&\text{CONSTRUCT}\;\; onto : \{\; ?\,Test(?\,Id)\;\} \\
&\text{WHERE}\;\{\; lab : ?\,Id\;?\,T\;?\,Test\;. \\
&\qquad\qquad \text{FILTER}\;(?\,T = \mathbf{xray} \vee ?\,T = \mathbf{blood})\;\} \\
r_2 = \;&\text{CONSTRUCT}\;\; dss : \{\; need(?\,Id, ab), forbid(?\,Id, ?\,Drug)\;\} \\
&\text{WHERE}\;\{\; onto : Pneumonia(?\,Id)\;. \\
&\qquad\qquad \text{OPT}\;\{\; patients : ?\,Id\;\langle\#allergy\rangle\;?\,Drug\;\}\;\} \\
r_3 = \;&\text{CONSTRUCT}\;\; dss : \{\; give(?\,Id, ab\_s)\;\} \\
&\text{WHERE}\;\{\; onto : AtypPneumonia(?\,Id)\;\}
\end{aligned}
$$

Bridge r_1 imports xray and blood test results from patients into the ontology. Bridge r_2 links pneumonia with a medication requirements (antibiotics) and optionally forbids drugs if an allergy is known. Bridge r_3 triggers administration of antibiotic 'ab_s' for atypical (severe form of) pneumonia. □

We next introduce the formalism that tackles the core challenge of variables in knowledge interlinking, then we formally introduce SPARQL-MCS syntax and semantics.

3.1 Abstract Variable Substitution

One of the strengths of MCSs is the abstract view on heterogeneous systems, where formulas and beliefs are unstructured objects. However, our SPARQL bridges use variables, therefore we need to establish a way how those variables interact with (probably variable-free) beliefs of contexts and formulas in knowledge bases.

Given an MCS M, we assume a set of variable symbols V which is common to all contexts; wlog. we assume $V = \{?X, ?Y, ?Z, \ldots\}$. With respect to a logic $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$, there are two kinds of expressions containing variables: (a) $\mathcal{F}$-expressions $VE^{\mathbf{KB}_L}$, which are knowledge base formulas containing variables, and (b) $\mathcal{B}$-expressions $VE^{\mathbf{BS}_L}$, which are beliefs containing variables. Intuitively, an expression is a formula or a belief of L containing variables of V.

Substituting all variables in an $\mathcal{F}$-expression ($\mathcal{B}$-expression) yields a knowledge base formula $s \in \bigcup \mathbf{KB}_L$ (a belief $b \in \bigcup \mathbf{BS}_L$). Given expression e we denote by $vars(e)$ the set of all variables that occur in e.

Example 5 (ctd). For $C_{patients}$ which uses L_{RDF}, '$?X\;\langle\#allergy\rangle\;?Y$' in $VE^{\mathbf{BS}_{RDF}}$ is used to query allergies of patients, while '$?X\;?Y\;?Z$' in $VE^{\mathbf{KB}_{RDF}}$ could be used to add arbitrary triples to $C_{patients}$.

For C_{onto}, '$Pneumonia(?\,Id)$' in $VE^{\mathbf{BS}_{\mathcal{AL}}}$ allows to query patients in the concept *Pneumonia*, and '$?\,Test(?\,Id)$' in $VE^{\mathbf{KB}_{\mathcal{AL}}}$ allows to assert membership of an individual $?\,Id$ in some concept $?\,Test$ in the ontology. □

As the shape of an object substituted for a variable depends on the specifics of the logic at hand, we introduce a finite, *abstract universe U* containing a

context-independent representation of all individuals that can be substituted for variables in the system. A *variable mapping* (for short, mapping) $\mu : V \to U$ is a partial function mapping variables to individuals of the abstract universe. Such μ then is used to substitute variables in an expression with their context-dependent representation. Abusing notation, we also write μ for a mapping of multiple distinct variables and denote its signature by $V \to U$.

Example 6 (ctd). We represent patient Sue as $\mathbf{sue} \in U$. Intuitively, applying mapping $\mu_{ex} = \{?X \mapsto \mathbf{sue}\}$ to an RDF-triplestore expression '$?X \; \langle \#ssn \rangle \; 123$' maps object $\mathbf{sue}$ into an IRI and yields '$\langle \#sue \rangle \; \langle \#ssn \rangle \; 123$'. Applying μ_{ex} to $\mathcal{AL}$ expression '$Pneumonia(?X)$' maps object $\mathbf{sue}$ into individual constant 'sue' and yields '$Pneumonia(sue)$'. □

In the following we introduce an abstraction for (i) applying a mapping to a $\mathcal{F}$-expression, which yields a formula without variables, and for (ii) matching a $\mathcal{B}$-expression with a belief set of a certain context, which yields all matching variable mappings.

Definition 1. *Given sets V, U, and $VE^{\mathbf{KB}_L}$ as described above, a variable substitution $subst_L$ is a function $subst_L : VE^{\mathbf{KB}_L} \times (V \to U) \to \bigcup \mathbf{KB}_L$ such that $subst_L(e, \mu) = e'$ iff applying μ to e yields e' and $e' \in \bigcup \mathbf{KB}_L$.*

Intuitively, $subst_L$ applies the mapping μ to a given $\mathcal{F}$-expression e and yields a knowledge base formula e'. Defining $subst_L$ for a logic L has the effect of defining what *'applying substitution μ to expression e'* means for formulas of L.

Example 7 (ctd). Reusing μ_{ex}, $subst_{RDF}(?X \; \langle \#allergy \rangle \; ab1, \mu_{ex}) = $ '$\langle \#sue \rangle \; \langle \#allergy \rangle \; ab1$' and $subst_{\mathcal{AL}}(Pneumonia(?X), \mu_{ex}) = $ '$Pneumonia(sue)$'. Therefore, in L_{RDF} entities are substituted as IRIs, in $L_{\mathcal{AL}}$ as individual symbols. □

Definition 2. *Given sets V, U, and $VE^{\mathbf{BS}_L}$ as described above, a variable matching $match_L$ is a function $match_L : VE^{\mathbf{BS}_L} \times \mathbf{BS}_L \to 2^{V \to U}$ such that $match_L(e, bs) = \{\mu \mid applying \; \mu : V \to U \; to \; e \; yields \; some \; e' \in bs\}$.*

Intuitively, $match_L$ takes a $\mathcal{B}$-expression e and returns all mappings that, applied to e, yield a belief of bs. Similarly as above, $match_L$ captures what 'applying μ to e' means for $\mathcal{B}$-expressions of L.

Example 8. Given belief set $bs = \{\langle \#sue \rangle \; \langle \#allergy \rangle \; ab1\} \in \mathbf{BS}_{RDF}$, we have $match_{RDF}(?X \; \langle \#allergy \rangle \; ab1, \; bs) = \{\mu_{ex}\}$ with μ_{ex} the only mapping. □

3.2 SPARQL-MCS Syntax

We modify SPARQL's CONSTRUCT$\{H\}$ WHERE$\{B\}$ queries to establish knowledge exchange between contexts. Instead of graph templates H we permit sets of $\mathcal{F}$-expressions that yield formulas which are compatible with the context which receives information from the query. Similarly, we extend the concept of graph pattern expressions B and SPARQL built-in conditions by allowing $\mathcal{B}$-expressions and elements of the universe U as basic building blocks. E.g., if a context is a

propositional logic, then the pattern expression of our bridge allows to query that context with any propositional formula occuring at those positions where in ordinary SPARQL queries an RDF triple occurs.

Definition 3. *A graph pattern expression (gpe) wrt. logics $L_1, \ldots, L_n$ is recursively defined as follows:*
$-$ *if p is a belief of logic L_i (there is a $bs \in \mathbf{BS}_i$ with $p \in bs$) then $(i:p)$ is a gpe,*
$-$ *if $p \in VE^{\mathbf{BS}_{L_i}}$ is a $\mathcal{B}$-expression of L_i, then $(i:p)$ is a gpe,*
$-$ *if P_1 and P_2 are gpes then so are $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ UNION } P_2)$,*
$-$ *if P is a gpe and R is a built-in condition (bic), then $(P \text{ FILTER } R)$ is a gpe.*

A bic is constructed using variables from V, objects from U, logical connectives $(\neg, \wedge, \vee)$, equality $(=)$, and the unary predicate 'bound':
$-$ *if $?X, ?Y \in V$ and $o \in U$, then $bound(?X)$, $?X = o$, and $?X = ?Y$ are bics, and*
$-$ *if R_1 and R_2 are bics, then $(\neg R_1)$, $(R_1 \vee R_2)$, and $(R_1 \wedge R_2)$ are bics.*

Definition 4. *An L_k-SPARQL bridge over logics $\mathcal{L} = (L_1, \ldots, L_n)$ is of the form*

$$\text{CONSTRUCT } k : \{H\} \text{ WHERE } \{P\}, \tag{1}$$

where $H \subseteq VE^{\mathbf{KB}_{L_k}}$, P is a gpe , and $1 \leq k \leq n$.

Definition 5. *A SPARQL-MCS $M = (C_1, \ldots, C_n)$ is a collection of contexts $C_i = (L_i, kb_i, sbr_i)$, $1 \leq i \leq n$, where $L_i = (\mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i)$ is a logic, $kb_i \in \mathbf{KB}_i$ a knowledge base, and sbr_i is a set of L_i-SPARQL bridges over $(L_1, \ldots, L_n)$.*

3.3 SPARQL-MCS Semantics

Semantics of SPARQL-MCS are defined in terms of equilibria, as for ordinary MCS, with the difference being in the notion of applicable bridges. Therefore we define semantics of SPARQL bridges, gearing the definitions to the SPARQL semantics in [2].

Analogous to applicable rules in ordinary MCSs, we first define how variable mappings are obtained from a SPARQL bridge and a belief state. We then describe how these variable mappings instantiate a set of CONSTRUCT templates in a SPARQL bridge. This set is then added to the respective knowledge bases and corresponds to the set of heads of applicable bridge rules in ordinary MCS.

For variable mappings the following operations are used: $dom(\mu)$ denotes the subset of V where the partial mapping μ is defined, furthermore we call two mappings μ_1 and μ_2 *compatible* iff for all $x \in dom(\mu_1) \cap dom(\mu_2)$ it holds that $\mu_1(x) = \mu_2(x)$.

We next define the semantics $[[\cdot]]_S$ of a gpe wrt. a belief state S in an MCS M. Note that the main difference to existing semantics of SPARQL are the first two cases.

Definition 6. *Let $S = (S_1, \ldots, S_n)$ be a belief state, $p \in \bigcup \mathbf{BS}_i$ a belief, $e \in VE^{\mathbf{BS}_i}$ a $\mathcal{B}$-expression, $1 \leq i \leq n$, P_1, P_2 gpes, and R a built-in condition (bic). Then $[[\cdot]]_S$ is recursively defined as follows:*

- $[[(i\!:\!p)]]_S = \{\emptyset\}$ *if* $p \in S_i$, $\emptyset$ *otherwise,*
- $[[(i\!:\!e)]]_S = match_{L_i}(e, S_i)$,
- $[[P_1 \text{ AND } P_2]]_S = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [[P_1]]_S \text{ and } \mu_2 \in [[P_2]]_S \text{ are compatible}\}$,
- $[[P_1 \text{ OPT } P_2]]_S = \{\mu \mid \mu \in [[P_1 \text{ AND } P_2]]_S \text{ or } \mu \in \Omega\}$ *where*
     $\Omega = \{\mu \in [[P_1]]_S \mid \text{ for all } \mu' \in [[P_2]]_S, \mu \text{ and } \mu' \text{ are not compatible}\}$,
- $[[P_1 \text{ UNION } P_2]]_S = \{\mu \mid \mu \in [[P_2]]_S \text{ or } \mu \in [[P_2]]_S\}$, *and*
- $[[P_1 \text{ FILTER } R]]_S = \{\mu \in [[P_1]]_S \mid \mu \models R\}$.

where $\mu \models R$ *holds (for variables* $?X, ?Y \in V$, *constant* $o \in U$, *and bics* R', R_1, R_2) *if* R *is of form*

- $bound(?X)$ *and* $?X \in dom(\mu)$,
- $?X = o$, $?X \in dom(\mu)$ *and* $\mu(?X) = o$,
- $?X = ?Y$, $?X \in dom(\mu), ?Y \in dom(\mu)$ *and* $\mu(?X) = \mu(?Y)$,
- $\neg(R')$ *and* $\mu \models R'$ *does not hold,*
- $(R_1 \vee R_2)$ *and at least one of* $\mu \models R_1$ *and* $\mu \models R_2$ *holds,*
- $(R_1 \wedge R_2)$ *and* $\mu \models R_1$ *and* $\mu \models R_2$.

For a bridge r of form (1), and a belief state S, the set of constructed formulas $constr_k(r, S) = \{e \mid e = subst_{L_k}(h, \mu), \ \mu \in [[P]]_S, \ h \in H\}$ contains all knowledge base formulas resulting from substituting a head $h \in H$ with a variable mapping obtained from evaluating P wrt. S.[2] Finally, the set of formulas constructed at context C_k wrt. a belief state S is defined as $app_k(S) = \{constr_k(r, S) \mid r \in sbr_k\}$.

Example 9 (ctd). Consider bridge r_2 of our running example, its query expressions evaluated against belief state S with $S_{patients} = \{\langle \#sue\rangle \ \langle \#allergy\rangle \ ab\_s. \}$ and $S_{onto} = \{Pneumonia(sue)\}$ yields a mapping $\mu = \{?Id \mapsto sue, ?Drug \mapsto ab\_s\}$. Similarly, for S' with $S'_{patients} = \emptyset$ and $S'_{onto} = S_{onto}$, mapping μ' is $\{?Id \mapsto sue\}$.

Now we have all necessary notions to define semantics of the whole system.

Definition 7. *A belief state* $S = (S_1, \ldots, S_n)$ *of an SPARQL-MCS* M *is an equilibrium iff for all* $1 \leq i \leq n$ *holds* $S_i \in ACC_i(kb_i \cup app_i(S))$.

Example 10 (ctd). Our running example has one equilibrium $S = (kb_{patients}, kb_{lab}, \{Pneumonia(sue)\}, \{need(sue, ab), forbid(sue, ab\_s), give(sue, ab2)\})$. □

4 Monotonicity and Minimal Equilibria

Minimal equilibria are of special interest as they precisely capture that knowledge which necessarily follows from a given SPARQL-MCS with no superflous information being present in such an equilibrium. Such a unique minimal equilibrium can be computed using the least fixed-point of an immediate-consequences operator.

This, however, only applies to monotonic SPARQL-MCS where all contexts and SPARQL bridges are *monotonic*. For SPARQL bridges, several options

---

[2] Note, that invalid formulas or formulas where not all variables were substituted, are simply ignored; furthermore if $[[P]]_S = \emptyset$, then $constr_k(r, S) = \emptyset$.

exist; e.g., if the OPT is disregard, the resulting bridges are monotonic. For a context C_i, it is monotonic iff i) for all belief states S holds that the semantics gives a unique result, i.e., $\mathbf{ACC}_i(kb_i \cup app_i(S)) = \{N\}$, and ii) this result is monotonic in S, i.e., for all belief states $S \subseteq S'$ with $\mathbf{ACC}_i(kb_i \cup app_i(S)) = \{N\}$ and $\mathbf{ACC}_i(kb_i \cup app_i(S)) = \{N'\}$ holds $N \subseteq N'$ where $S \subseteq S'$ holds if $S = (S_1, \ldots, S_n), S' = (S'_1, \ldots, S'_n)$ and for all $1 \leq i \leq n$ holds $S_i \subseteq S'_i$.

We give the immediate-consequences operator on monotonic SPARQL-MCS:

Definition 8. *Let M be an MCS and $\mathbf{S}$ the set of all belief states of M, the immediate-consequences operator $T_M : \mathbf{S} \to \mathbf{S}$ is defined as $T_M(S) = (S'_1, \ldots, S'_n)$ where for all $1 \leq i \leq n$ we have $ACC_i(kb_i \cup app_i(S)) = \{S'_i\}$.*

One can show that the following holds:

Proposition 1. *Let M be a monotonic SPARQL-MCS and T_M the respective immediate-consequences operator, then T_M is monotonic and continous.*

Using the Knaster-Tarski theorem we therefore conclude that the minimal equilibrium S_m of M equals the least-fixed point of T_M, i.e.,

$$S_m = lfp(T_M) = T_M(T_M(\ldots T_M((\emptyset_1, \ldots, \emptyset_n)) \ldots)).$$

As our abstract universe U and the set of bridge queries are finite, it follows that each set of constructed formulas, $app_i(S)$, for any belief state S and context C_i is finite. Therefore the least-fixed point of T_M is reached after finitely many steps.

For non-monotonic SPARQL-MCS (by non-monotonic contexts or unrestricted SPARQL bridges), the problem of finding an equilibrium becomes harder as recursive negation comes into play (see also [11]) as there is a significant increase in computational complexity even for bridge rules without variables (cf. [4]).

5 Related and Future Work

An existing proposal to support reasoning with variables in bridge rules of ordinary MCSs extends logics in a bottom-up way to 'relational logics' [8] by adding an extra set of predicate symbols and variables that can be used in bridge rules. Different from that, our approach is top-down as we introduce an intermediate variable mapping layer, which does not impose any restrictions on a context's logic or syntax. We also extend bridge rules, allowing for more powerful queries via the OPT and FILTER conditions.

Note that the extension of [8] closely follows the way variables are treated in logic programming where the task of instantiating (grounding) a logic program with its universe of constants is a problem that is very relevant for the performance of current solver engines. In this work, we disregard this issue and focus on the formalism of substitution itself. Nevertheless, grounding strategies, e.g., [10], are highly relevant for implementing SPARQL-MCS.

The MWeb approach [1] also is related to this work. It links heterogeneous knowledge-based systems with rules that may contain variables. Different from

this approach, MWeb restricts context logics (there called constituent rule bases) and does not consider converting between constants in different constituent rule bases.

Another approach for extending RDF and SPARQL with a notion of context is the N-Quads [6] proposal. For provenance, RDF triples are extended to quadruples containing an additional identifier marking the origin of the RDF triple. The N-Quads proposal does not consider heterogeneous knowledge sources, but the SPARQL-MCS framework is general enough to support contexts and SPARQL bridges using such N-Quads.

For future work we plan to investigate the complexity of SPARQL-MCS and optimised evaluation algorithms for SPARQL bridges. We also aim for supporting value invention across heterogeneous knowledge sources to allow blank nodes in construct templates.

References

1. Analyti, A., Antoniou, G., Damasio, C.V.: MWeb: A principled framework for modular web rule bases and its semantics. ACM Trans. Comput. Logic 12(2) (2011)
2. Arenas, M., Gutierrez, C., Pérez, J.: Foundations of RDF Databases, pp. 158–204. Springer (2009)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
4. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI. pp. 385–390 (2007)
5. Brewka, G., Roelofsen, F., Serafini, L.: Contextual default reasoning. In: IJCAI. pp. 268–273 (2007)
6. Cyganiak, R., Harth, A., Hogan, A.: N-Quads: Extending N-Triples with Context (2009), http://sw.deri.org/2008/07/n-quads/
7. Eiter, T., Fink, M., Weinzierl, A.: Preference-based inconsistency assessment in multi-context systems. In: JELIA (2010)
8. Fink, M., Ghionna, L., Weinzierl, A.: Relational information exchange and aggregation in multi-context systems. In: LPNMR (2011), to appear
9. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics, or: How we can do without modal logics. Artificial Intelligence 65(1), 29–70 (1994)
10. Leone, N., Perri, S., Scarcello, F.: Improving ASP instantiators by join-ordering methods. In: LPNMR, pp. 280–294 (2001)
11. Polleres, A.: From sparql to rules (and back). In: Williamson, C.L., Zurko, M.E., Patel-Schneider, P.F., Shenoy, P.J. (eds.) WWW. pp. 787–796. ACM (2007)
12. Roelofsen, F., Serafini, L.: Minimal and absent information in contexts. In: IJCAI. pp. 558–563 (2005)
13. Serafini, L., Giunchiglia, F., Mylopoulos, J., Bernstein, P.A.: Local relational model: A logical formalization of database coordination. In: CONTEXT. pp. 286–299 (2003)