A semantic stateless service description language

P. A. Bonatti and L. Sauro

Università di Napoli Federico II

Abstract. Complexity issues and the requirements on semantic web application in the Life Science domains recently motivated a few works on stateless service description languages [1, 5]. With stateless services, it is possible to reason about the semantic relationships between inputs and outputs, while keeping matchmaking and composition decidable. In this paper we extend the languages introduced in [1] and [5] with more general forms of composition and other constructs. We provide formal syntax and semantics and some preliminary results on the complexity of service comparison. These complexity results rely on hybrid formalisms involving both logic programming rules and description logics.

1 Introduction

The area of semantic web services is concerned with the declarative, knowledge based specification of web service semantics applied to service matchmaking (i.e., finding a service that matches a given specification), verification and automated composition. There is a conspicuous literature on the topic, enriched by several competing standards, such as OWL-S, WSMO, and WSDL-S.

When the semantic description involves dynamic behavioral aspects such as iterations, the tasks of matchmaking and composition easily become undecidable. This motivated a few works on stateless services [1, 5], that behave like functions or database queries. With stateless services, it is possible to move beyond a mere description of input and output types and capture the *relationships* between inputs and outputs, while keeping matchmaking and composition decidable. Stateless services are interesting because they are common in the domain of Life Sciences [5]. Moreover, they can be paired with a workflow language supporting procedural constructs like BPEL4WS with the purpose of supporting the dynamic binding of atomic activities.

In this paper we extend the languages introduced in [1] and [5] with more general forms of composition and other constructs. We provide formal syntax and semantics and some preliminary results on the complexity of service comparison, a basic reasoning task that underlies both matchmaking and composition (cf. [1]). These complexity results rely on hybrid formalisms involving both rules and description logics. The language we adopt admits a graphic presentation (that may be appreciated by users with limited programming skills) as well as textual representation that resembles relational query and programming languages enough to be familiar to programmers.

We start with some examples (Sec. 2) followed by a brief summary of description logic notions (Sec. 3). Then we formalize our service description logic language $\mathcal{SDL}_{full}$ (Sec. 4). Service comparison is reduced to an intermediate logic programming formulation and then to queries against description logic knowledge bases in Sec. 5, which allow to derive complexity results (Sec.6).

2 A running example

Services receive input messages and return output messages. Such messages are structured objects (as in WSDL), consisting of a set of attribute-value pairs, such as

{`street="Via Toledo", numb=128`}.

Following [1], we assume that services can be like queries, that is, a single input message may be mapped onto a *set* of homogeneously structured output messages. Formally this means that a service can be abstracted by any set of pairs $(m_{\mathsf{in}}, m_{\mathsf{out}})$, with multiple pairs sharing the same m_{in}.

Now assume an underlying ontology defines the concepts `Place`, `Map`, `Coord`, and `Address`, and that every `Place` has the attributes `hasAddr`, `hasMap`, `hasCoord`. In turn each address has the attributes `hasCity`, `hasStr` and `hasNum`. Consider a service *Mapservice* that takes input messages with attributes `city` and `street`, and returns the map of the surrounding area in a message with the single field `result`. *Mapservice* can be described in our language with the following expression:

```
select result:=hasMap from all Place
    with hasAddr.hasCity = city, hasAddr.hasStr = street .
```

This description can be easily adapted to describe similar services. For example, a specialized map service that works only for southern cities can be described by defining a concept `SouthernCity` in the underlying ontology and restricting *Mapservice* with the expression:

Mapservice `restricted to SouthernCity(city) .`

Portals can be described with unions. Given two map services for Europe and China, called *Euromap* and *Chinamap*, a portal that covers both areas can be described by:

`union` (*Euromap, Chinamap*).

Intersections are supported, too. Now suppose that *Euromap* is more reliable than the generic *Mapservice*, then it may be preferable to use *Euromap* when possible. This can be done with conditionals (temporarily assume that *Euromap* and *Chinamap* have the same input message type as *Mapservice*, with the `city` field):

`if EuropeanCity(city) then` *Euromap* `else` *Mapservice* .

A relevant task is composition, our framework supports composition through *dataflow graphs* by which the output of some services can be fed as input to other services. For example, let *Addr2coor* be a service that takes `city` and `street` and returns the associated coordinates `lat` and `lon`; then let *Coor2map* be a service that returns the map associated to the given coordinates, called `latitude` and `latitude` by this service. The composition of these two services can be specified with the dataflow graph in Fig. 1. We support also a textual representation:

CompoundMap:

```
    in city, street
    out result
    C := Addr2coor(in)
    out := Coor2map(latitude:=C.lat, longitude:=C.lon).
```

In order to combine different services it may be necessary to adapt and restructure their inputs and outputs (e.g. consider the above example for conditionals when *Euromap* and *Chinamap* have different input message types). Here is an example of a variant of

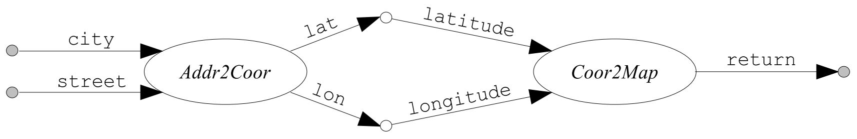

Fig. 1. A dataflow graph

Mapservice whose input city is forced to be `Naples` (a constant in the knowledge base), and whose output is renamed:

```
RestructuredMapServ:
    in street
    out map
    C := Mapservice(street:=in.street, city:=Naples)
    out.map:=C.result.
```

In general we allow a message element to be fed as an input to multiple other services, so dataflow graphs can be arbitrary DAGs. This was not allowed in [1]

Our framework allows to reason about different specifications. The basic reasoning task is *service comparison*, that given two service descriptions S_1 and S_2 checks whether all the input-output message pairs in the semantics of S_1 are also in the semantics of S_2; in that case we write $S_1 \sqsubseteq_{KB,\Sigma} S_2$, where KB is the underlying ontology and Σ contains the service definitions. By comparing services one may look for stronger or weaker services (cf. [1]). If *Addr2coor* and *Coor2map* are correctly specified (say, with `select` expressions), then our framework can verify that *CompoundMap* $\sqsubseteq_{KB,\Sigma}$ *Mapservice* and *Mapservice* $\sqsubseteq_{KB,\Sigma}$ *CompoundMap*, thereby concluding that in the absence of a direct implementation of *Mapservice*, an equivalent service can be obtained by composing the implementations of *Addr2coor* and *Coor2map* as specified by *CompoundMap* (dynamic service replacement). Service comparison can also be a basis for automated composition that, however, lies beyond the scope of this paper.

Syntactically speaking, the service description language illustrated above lies somewhere in between relational algebra and a programming language. A major difference with respect to both is that descriptions are linked to an ontology, so it is possible to distinguish—say—a hash table that associates people with their age from another hash table (with the same implementation) that associates people with their credit card number. Clearly, such differences are crucial for tasks such as service discovery and dynamic binding of workflow activities to services. Procedural constructs cover assignments and conditionals; only iterations are not supported, and this has a few advantages: (i) the main reasoning tasks are decidable, (ii) the language is easier to use for people with no programming background.

3 Preliminaries

The vocabulary of the description logics we deal with in this paper consists of the following pairwise disjoint countable sets of symbols: a set of *atomic concepts* At, a set of individual names In, and a set of *atomic roles* $\mathsf{A_R}$, with a distinguished subset of names $\mathsf{A_{tR}} \subseteq \mathsf{A_R}$ denoting *transitive roles*.

A *role* is either an expression P or P^-, where $P \in \mathsf{A_R}$. Let R range over roles. The set of *concepts* is the smallest superset of At such that if C, D are concepts, then $\top$, $\neg C$, $C \sqcap D$, $\exists.C$, and $\exists^{\leq n} R.C$ are concepts.

Semantics is based on interpretations of the form $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}}$ is a set of *individuals* and $\cdot^{\mathcal{I}}$ is an interpretation function mapping each $A \in \mathsf{At}$ on some $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each $a \in \mathsf{In}$ on some $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, and each $R \in \mathsf{A_R}$ on some $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Moreover, if $R \in \mathsf{A_{tR}}$, then $R^{\mathcal{I}}$ is transitive. The meaning of inverse roles is $(R^-)^{\mathcal{I}} = \{\langle y, x \rangle \mid \langle x, y \rangle \in R^{\mathcal{I}}\}$. Next we define the meaning of compound concepts. By $\sharp\, S$ we denote the cardinality of S.

$$\begin{aligned}
A^{\mathcal{I}}_{\rho} &= A^{\mathcal{I}} \qquad (A \in \mathsf{At}) \qquad \top^{\mathcal{I}} = \Delta^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}}_{\rho} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}_{\rho} \qquad\qquad (C \sqcap D)^{\mathcal{I}}_{\rho} = C^{\mathcal{I}}_{\rho} \cap D^{\mathcal{I}}_{\rho} \\
(\exists R.C)^{\mathcal{I}}_{\rho} &= \left\{ x \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}_{\rho} \right\} \\
(\exists^{\leq n} R.C)^{\mathcal{I}}_{\rho} &= \left\{ x \mid \sharp\{ y \mid \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}_{\rho} \} \leq n \right\} .
\end{aligned}$$

Other standard constructs ($\forall R.C,\ \bot, \sqcup$) can be derived from the above concepts.

A *general concept inclusion* (GCI) is an expression $C \sqsubseteq D$ where C and D are concepts. A *role inclusion* is an expression $R_1 \sqsubseteq R_2$ where R_1 and R_2 are roles. An *assertion* is an atom like $A(a)$ or $P(a, b)$ where $A \in \mathsf{At}$, $R \in \mathsf{A_R}$, and $\{a, b\} \in \mathsf{In}$. A *TBox* is a set of GCIs; a *role hierarchy* is a set of role inclusions; an *ABox* is a set of assertions. Finally, a DL knowledge base (DL KB) is a triple $\langle \mathcal{T}, \mathcal{H}, \mathcal{A} \rangle$ consisting of a TBox, a role hierarchy and an ABox.

An interpretation $\mathcal{I}$ satisfies a (concept or role) inclusion $E_1 \sqsubseteq E_2$ iff $E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}}$. Moreover $\mathcal{I}$ satisfies an assertion $A(a)$ (resp. $P(a, b)$) iff $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp. $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$). A *model* of a DL KB is any $\mathcal{I}$ that satisfies all the inclusions and the assertions of the KB.

The above description logic is known as $\mathcal{SHIQ}_{R+}$. By disallowing transitive roles we get $\mathcal{SHIQ}$. By disallowing $\exists^{\leq n} R.C$, transitive roles and role hierarchies one gets the logic $\mathcal{ALCI}$. $\mathcal{ALC}$ is obtained by further dropping inverse roles. The logic $\mathcal{EL}$ supports only $\sqcap$, $\exists R.\top$, and GCIs built from these constructs.

Moreover, there exists a rather different extension of $\mathcal{ALC}$ called $\mathcal{DLR}$, supporting n-ary relations ($n > 2$) that we will mention in the following but we do not report here due to space limitations. Its definition and relevant results can be found in [3].

4 Syntax and semantics of $\mathcal{SDL}_{full}$

Our service description language, called $\mathcal{SDL}_{full}$, extends the DL vocabulary with an infinite supply of constants $\mathsf{N_c}$, service names $\mathsf{N_s}$, and message attribute names $\mathsf{N_a}$. $\mathcal{SDL}_{full}$ describes functional and knowledge-based aspects of web-services. Therefore, as usual functional programming languages, it does not define a service as a set of state variables and a sequence of statements which update them, but as the functional composition of stateless expressions that have to be evaluated.

Definition 1. *The language of* service expressions *is the least set* Expr *containing:*

- *(service calls) all $S \in \mathsf{N_s}$;*
- *(set operators) all expressions $op(E_1, \ldots, E_n)$ such that $\{E_1, \ldots, E_n\} \subseteq$* Expr *and $op \in \{\mathtt{union}, \mathtt{intersection}\}$;*

– *(conditionals) all expressions* $\texttt{if}\, L\, \texttt{then}\, E_1 [\texttt{else}\, E_2]$ *(the else clause is optional) such that*
  - $\{E_1, E_2\} \subseteq \mathsf{Expr}$*, and*
  - *L is a list of* conditions *of the form $t = u$, $t \neq u$, $A(t)$, or $\neg A(t)$, where* $\{t, u\} \subseteq \mathsf{N_c} \cup \mathsf{N_a}$

– *(selections) all expressions* $\texttt{select}\, a_1 := r_1, \dots a_n := r_n\, \texttt{from all}\, D\, \texttt{with}\, L$*, such that*
  - $a_i \in \mathsf{N_a}$ $(1 \leq i \leq n)$*;*
  - *r_i is a role path (in the language of the underlying ontology)* $(1 \leq i \leq n)$*;*
  - *D is a concept (in the language of the underlying ontology);*
  - *L is a list of* bindings *$p_i = t_i$ $(1 \leq i \leq m)$ where each p_i is a role path (in the language of the underlying ontology), and $t_i \in \mathsf{N_c} \cup \mathsf{N_a}$;*

– *(message restructuring) all expressions $a_1 := t_1, \dots a_n := t_n$ such that $a_i \in \mathsf{N_a}$ and $t_i \in \mathsf{N_a} \cup \mathsf{N_c}$ $(1 \leq i \leq n)$;*

– *(restrictions) all expressions* $E\, \texttt{restricted to}\, L$ *such that L is a list of conditions (see conditionals above).*

A service consists of a dataflow graph which *evaluates* data by means of *functional* nodes. Each functional node represents a stateless expression which may have multiple inputs and outputs denoted by parameter names. Edges in a dataflow graph are used to connect the output of a functional node with the inputs of (possibly many) other functional nodes. In order to specify which output is connected to which input, edges are also labeled with attribute names and, as the inputs and the outputs of different expressions may be labeled with different parameter names, edges do not connect directly two functional nodes, but connect functional nodes with *parameter* nodes that are intended to fix name mismatches.

Dataflow graphs are defined as follows:

Definition 2. *A* dataflow graph *with name S is a tuple $\langle S, N_S, E_S, name_S, expr_S \rangle$ where*

– *$S \in \mathsf{N_s}$;*
– *N_S is a finite set of nodes, partitioned into* functional *and* parameter *nodes, denoted by $\mathsf{fun}(N_S)$ and $\mathsf{par}(N_S)$, respectively;*
– *E_S is a finite set of edges; $E_S \subseteq (\mathsf{fun}(N_S) \times \mathsf{par}(N_S)) \cup (\mathsf{par}(N_S) \times \mathsf{fun}(N_S))$;*
– *$name_S : \mathsf{par}(N_S) \cup E_S \to \mathsf{N_a}$ is a labelling function;*
– *$expr_S : \mathsf{fun}(N_S) \to \mathsf{Expr}$ is a labelling function.*

Moreover, dataflow graphs are required to be directed acyclic graphs (DAGs).

The parameter nodes with no incoming edges (resp. no outgoing edges) will be called the *input nodes* (resp. *output nodes*) of the graph. In Fig. 1, ovals and small circles represent functional and parameter nodes, respectively; input and output nodes are colored in gray.

The *dependency graph* of a set of dataflow graphs Σ is $\langle \Sigma, E \rangle$, where E is the set of all pairs (G_1, G_2) such that the name of G_2 occurs in the label of some functional node in G_1. We say that Σ is *acyclic* if its dependency graph is.

Definition 3. *A* service specification *Σ is a finite, acyclic set of dataflow graphs with mutually different names.*

Edge labels should match the input/output message attributes of the service expressions labeling functional nodes. This requirement is formalized in terms of typing. In this paper we only deal with a structural form of typing (centred around message attribute names); the problem of ensuring–say—that the connected input/output attributes `lat` and `latitude` in Fig. 1 belong respectively to two "compatible" concepts C_1 and C_2 such that $C_1 \sqsubseteq C_2$ has already been tackled in the literature (including [5]). We will deal with it in the full paper.

Definition 4. *A (message)* type *is a finite set* $T \subseteq \mathsf{N_a}$

Definition 5. *The* input type of a dataflow graph $G = \langle S, N_S, E_S, name_S, expr_S\rangle$ *with respect to a specification* Σ *is the set*

$$\mathsf{in}_\Sigma(G) = \{name_S(n) \mid n \textit{ is an input node of } G\}\,.$$

The output type of a dataflow graph $G = \langle S, N_S, E_S, name_S, expr_S\rangle$ *with respect to a specification* Σ *is the set*

$$\mathsf{out}_\Sigma(G) = \{name_S(n) \mid n \textit{ is an output node of } G\}\,.$$

Definition 6. *The* input type of a service expression E *with respect to a specification* Σ*, denoted by* $\mathsf{in}_\Sigma(E)$*, is recursively specified as follows:*

- *if* $E = S \in \mathsf{N_s}$*, then* $\mathsf{in}_\Sigma(E)$ *equals* $\mathsf{in}_\Sigma(G)$ *where* G *has name* S*;*
- $\mathsf{in}_\Sigma(op(E_1, E_2)) = \mathsf{in}_\Sigma(E_1) \cup \mathsf{in}_\Sigma(E_2)$*;*
- $\mathsf{in}_\Sigma(\texttt{if}\, C\, \texttt{then}\, E_1\, \texttt{else}\, E_2) = \mathsf{in}_\Sigma(E_1) \cup \mathsf{in}_\Sigma(E_2) \cup \{a \in \mathsf{N_a} \mid a \textit{ occurs in } C\}$*;*
- $\mathsf{in}_\Sigma(\texttt{select}\, A\, \texttt{from all}\, D\, \texttt{with}\, R) = \{a \in \mathsf{N_a} \mid a \textit{ occurs in } R\}$*;*
- $\mathsf{in}_\Sigma(a_1 := t_1, \ldots a_n := t_n) = \mathsf{N_a} \cap \{t_1, \ldots t_n\}$.

The output type of a service expression E *with respect to a specification* Σ*, denoted by* $\mathsf{out}_\Sigma(E)$*, is recursively specified as follows:*

- *if* $E = S \in \mathsf{N_s}$*, then* $\mathsf{out}_\Sigma(E)$ *equals* $\mathsf{out}_\Sigma(G)$ *where* G *has name* S*;*
- $\mathsf{out}_\Sigma(op(E_1, E_2)) = \mathsf{out}_\Sigma(E_1) \cap \mathsf{out}_\Sigma(E_2)$*;*
- $\mathsf{out}_\Sigma(\texttt{if}\, C\, \texttt{then}\, E_1\, \texttt{else}\, E_2) = \mathsf{out}_\Sigma(E_1) \cap \mathsf{out}_\Sigma(E_2)$*;*
- $\mathsf{out}_\Sigma(\texttt{select}\, a_1 := r_1, \ldots a_n := r_n\, \texttt{from all}\, D\, \texttt{with}\, R) = \{a_1, \ldots a_n\}$*;*
- $\mathsf{out}_\Sigma(a_1 := t_1, \ldots a_n := t_n) = \{a_1, \ldots a_n\}$.

About the above definition: Intuitively, *all* input parameters have to be supplied in order to call a service; therefore if the components of a compound service have different input types, then the compound service must take their *union* to be sure that all component services can be invoked. Symmetrically, the only outputs one can count on are those returned by all the component services; this is why intersection is used here.

Definition 7. *A specification* Σ *is* well-typed *iff for all dataflow graphs* $\langle S, N_S, E_S, name_S, expr_S\rangle \in \Sigma$*, and for all functional nodes* $k \in \mathsf{fun}(N_S)$*,*

- $\mathsf{in}(k)$ *equals the set of labels of the incoming edges of* k*;*
- $\mathsf{out}(k)$ *contains the set of labels of the outgoing edges of* k*.*

From now on we assume that all service specifications are well-typed unless stated otherwise.

The semantics of service expressions and dataflow graphs is defined in terms of *worlds* that specify the extension of concepts and roles, as well as the behavior of each service. From a semantic perspective, a message is a partial function defined over the message's attributes, that returns for each attribute its value.

Definition 8. *A Δ-*message *is a partial function $m : \mathsf{N_a} \to \Delta$.*

The message's range Δ will sometimes be omitted when irrelevant or obvious.

Now a world is simply a combination of a DL interpretation (that interprets the terms defined in the underlying ontology) plus an interpretation of service names (i.e. atomic services).

Definition 9. *A* world *is a tuple $\mathcal{W} = \langle \Delta^{\mathcal{W}}, \cdot^{\mathcal{W}}, [\![\cdot]\!]^{\mathcal{W}} \rangle$ such that*

- *$\langle \Delta^{\mathcal{W}}, \cdot^{\mathcal{W}} \rangle$ is an interpretation of the knowledge base;*
- *$[\![\cdot]\!]^{\mathcal{W}}$ maps every service name $S \in \mathsf{N_s}$ on a set $[\![S]\!]^{\mathcal{W}}$ of $\Delta^{\mathcal{W}}$-message pairs.*

To ensure that service name evaluation reflects the given service specification, we have to specify the semantics of the terms and expressions used in dataflow graph labels.

Definition 10. *The* evaluation *$t^{\mathcal{W}}(m)$ of a* term *$t \in \mathsf{N_c} \cup \mathsf{N_a}$ with respect to a world $\mathcal{W}$ and a message m, is $m(t)$ if $t \in \mathsf{N_a}$, and $t^{\mathcal{W}}$ otherwise.*

Definition 11. *The* evaluation *$E^{\mathcal{W}}(m)$ of a* service expression *E with respect to a world $\mathcal{W}$ and a message m is recursively defined as follows:*

- *if $E = S \in \mathsf{N_s}$, then $E^{\mathcal{W}}(m) = \{m' \mid (m, m') \in [\![S]\!]^{\mathcal{W}}\}$;*
- $\mathtt{union}(E_1, E_2)^{\mathcal{W}}(m) = E_1^{\mathcal{W}}(m) \cup E_2^{\mathcal{W}}(m)$*;*
- $\mathtt{intersection}(E_1, E_2)^{\mathcal{W}}(m) = E_1^{\mathcal{W}}(m) \cap E_2^{\mathcal{W}}(m)$*;*
- $(\mathtt{if}\, C\, \mathtt{then}\, E_1\, \mathtt{else}\, E_2)^{\mathcal{W}}(m) = E_1^{\mathcal{W}}(m)$ *if $C^{\mathcal{W}}(m)$ is true, $E_2^{\mathcal{W}}(m)$ otherwise; moreover, $C^{\mathcal{W}}(m)$ is true iff*
  - *for all $t \odot u$ in C, $t^{\mathcal{W}}(m) \odot u^{\mathcal{W}}(m)$ holds ($\odot \in \{=, \neq\}$),*
  - *and for all literals $A(t)$ and $\neg B(u)$ in C, $t^{\mathcal{W}}(m) \in A^{\mathcal{W}}$ and $u^{\mathcal{W}}(m) \notin B^{\mathcal{W}}$;*
- $(\mathtt{select}\, a_1 := r_1, \ldots a_n := r_n\, \mathtt{from\ all}\, D\, \mathtt{with}\, R)^{\mathcal{W}}(m)$ *is the set of all m' such that, for some $x \in D^{\mathcal{W}}$,*
  - *for all $r \odot t$ in R there exists $y \in r^{\mathcal{W}}(x)$ such that $y \odot t^{\mathcal{W}}(m)$ holds ($\odot \in \{=, \neq\}$);*
  - *$m'(a_i) \in r_i^{\mathcal{W}}(x)$ $(1 \leq i \leq n)$; m' is undefined in every other case;*
- $(a_1 := t_1, \ldots a_n := t_n)^{\mathcal{W}}(m) = \{m'\}$ *where the domain of m' is $a_1, \ldots a_n$ and $m'(a_i) = m(t_i)$ $(1 \leq i \leq n)$.*

The evaluation of service compositions (i.e. dataflow graphs) is defined in a declarative way: each parameter node must be assigned a value (an element of $\Delta^{\mathcal{W}}$) in a way that is compatible with the input-output behavior of each functional node:

Definition 12. *The evaluation $[\![G]\!]^{\mathcal{W}}$ of a graph $G = \langle S, N_S, E_S, name_S, expr_S \rangle$ w.r.t. $\mathcal{W}$ is the set of all $\Delta^{\mathcal{W}}$-message pairs (m_{in}, m_{out}) such that for some function $\sigma : \mathsf{par}(N_S) \to \Delta^{\mathcal{W}}$, the following conditions hold:*

– *for all input nodes $n \in N_S$, $m_{in}(name_S(n)) = \sigma(n)$;*
– *for all output nodes $n \in N_S$, $m_{out}(name_S(n)) = \sigma(n)$;*
– *m_{in} and m_{out} are undefined for every other attribute name;*
– *for all $n \in \mathsf{fun}(N_S)$, it must hold that $m^n_{out} \in expr_S(n)^{\mathcal{W}}(m^n_{in})$, where m^n_{in} and m^n_{out} are defined as follows: for all $a \in \mathsf{N_a}$,*
  - *if there exists an edge (n', n) with $name_S(n', n) = a$, let $m^n_{in}(a) = \sigma(n')$,*
  - *if there exists an edge (n, n'') with $name_S(n, n'') = a$, let $m^n_{out}(a) = \sigma(n'')$,*
  - *m^n_{in} and m^n_{out} are undefined for all other inputs.*

Definition 13. *A world $\mathcal{W}$ is a* model *of a specification Σ with respect to a knowledge base KB iff*

1. *$\langle \Delta^{\mathcal{W}}, \cdot^{\mathcal{W}} \rangle$ is a model of KB;*
2. *for all names S of a dataflow graphs $G \in \Sigma$, $[\![S]\!]^{\mathcal{W}} = [\![G]\!]^{\mathcal{W}}$.*

If $\mathcal{W}$ is a model of Σ, then it is not hard to see that since Σ is acyclic (by definition), $[\![\cdot]\!]^{\mathcal{W}}$ is uniquely determined by $\langle \Delta^{\mathcal{W}}, \cdot^{\mathcal{W}} \rangle$ (i.e. service specifications are deterministic).

The next definition specifies when a service S_1 is a *weakening* of S_2 (equivalently, S_2 is a *strengthening* of S_1) [1]. These relations are the basis for service comparison.

Definition 14. *$S_1 \sqsubseteq_{KB,\Sigma} S_2$ iff for all models $\mathcal{W}$ of Σ w.r.t. KB, $[\![S_1]\!]^{\mathcal{W}} \subseteq [\![S_2]\!]^{\mathcal{W}}$.*

Roughly speaking, if S_2 is a strengthening of S_1, then for any given input, S_2 returns more answers than S_1. See [1] for a discussion of the different applications of strengthening and weakening in our reference scenarios.

5 Service comparison

Definition 15. *The* service comparison problem *is defined as follows: given KB, Σ, and two service names S_1 and S_2, decide whether $S_1 \sqsubseteq_{KB,\Sigma} S_2$.*

By translating service specifications into logic programming rules, service subsumption checking can be reduced to containment of unions of conjunctive queries (UCQ) against DL knowledge bases. In turn, this problem can be reduced to the evaluation of UCQs against DL knowledge bases.

5.1 Rules and queries

Consider rules like $A \leftarrow L_1, \ldots, L_n$ where A is a logical atom, each L_i is a literal (i.e. either an atom or a negated atom), possibly of the form $t = u$ or $t \neq u$. As usual, let $\mathsf{head}(r) = A$ and $\mathsf{body}(r) = \{L_1, \ldots, L_n\}$. *We restrict our attention to function-free rules only*: terms will be restricted to constants in In and variables.

The predicates in $\mathsf{body}(r)$ may be defined in a DL knowledge base, i.e. unary and binary predicates may belong to At and $\mathsf{A_R}$, respectively. If *all* the predicates occurring in $\mathsf{body}(r)$ belong to At and $\mathsf{A_R}$ and $\mathsf{body}(r)$ contains no occurrences of $\neg$, then we call r a *conjunctive query* (CQ). A *union of conjunctive queries* (UCQ) is a set of CQs having the same predicate name in the head. We add superscripts $\neq$, $\neg$ if the corresponding symbol may occur in $\mathsf{body}(r)$; for example UCQ$^{\neg}$ denotes the unions of conjunctive queries that may contain negative literals in the body.

Let $\mathcal{P}$ be a set of rules and $\mathcal{I}$ be an interpretation. Let an *$\mathcal{I}$-substitution* be a substitution that replaces each constant a by $a^{\mathcal{I}}$, and each variable with an element of $\Delta^{\mathcal{I}}$. $\mathcal{I}$-substitutions are a useful tool for defining the semantics of rules and queries.

Usually queries are evaluated against a knowledge base, and the answer is restricted to the individual constants that explicitly occur in the ABox (e.g. see [8]). In particular, a tuple $\boldsymbol{c}$ of constants is a *certain answer* of a CQ r against a DL KB $\mathcal{K}$ iff

- the constants in $\boldsymbol{c}$ occur in $\mathcal{K}$; moreover, for some substitution σ defined on the variables of $\mathsf{head}(r)$,
- $\mathsf{head}(r\sigma)$ has the form $p(\boldsymbol{c})$;
- for all models $\mathcal{I}$ of $\mathcal{K}$, there exists an $\mathcal{I}$-substitution θ such that every literal in $\mathsf{body}(r\sigma\theta)$ is *satisfied* by $\mathcal{I}$, that is,
  - for all $A(d)$ (resp. $\neg A(d)$) in $\mathsf{body}(r'\sigma\theta)$, $d \in A^{\mathcal{I}}$ (resp. $d \notin A^{\mathcal{I}}$) ;
  - for all $P(d,e)$ (resp. $\neg P(d,e)$) in $\mathsf{body}(r'\sigma\theta)$, $(d,e) \in P^{\mathcal{I}}$ (resp. $(d,e) \notin P^{\mathcal{I}}$) ;
  - all literals $d = e$ and $d \neq e$ in $\mathsf{body}(r'\sigma\theta)$ are true.

The set of all certain answers of a CQ r against $\mathcal{K}$ will be denoted by $\mathsf{c\_ans}(r,\mathcal{K})$. For a UCQ Q, let $\mathsf{c\_ans}(r,\mathcal{K}) = \bigcup_{r\in Q} \mathsf{c\_ans}(r,\mathcal{K})$.

In this paper, we will also query the *models* of a knowledge base and introduce what we call *unrestricted answers*, that are built from the domain elements of the models.[1] This definition applies to all sets of rules (not only CQs and UCQs).

The *$\mathcal{I}$-reduct of $\mathcal{P}$* , $\mathcal{P}^{\mathcal{I}}$, is the set of all rules r such that for some $r' \in \mathcal{P}$ and some $\mathcal{I}$-substitution σ,

- all literals belonging to $\mathsf{body}(r'\sigma)$ whose predicate is in $\mathsf{At} \cup \mathsf{A_R} \cup \{=,\neq\}$ are satisfied by $\mathcal{I}$;
- r is obtained from $r'\sigma$ by removing from $\mathsf{body}(r'\sigma)$ all the literals whose predicate is in $\mathsf{At} \cup \mathsf{A_R} \cup \{=,\neq\}$.

Note that the $\mathcal{I}$-reduct of a UCQ is always a set of facts.

We will denote by $\mathsf{lm}(\mathcal{P}^{\mathcal{I}})$ the least Herbrand model of $\mathcal{P}^{\mathcal{I}}$. The *unrestricted answer to a predicate p in $\mathcal{P}$* against $\mathcal{I}$ is $\mathsf{u\_ans}(p,\mathcal{P},\mathcal{I}) = \{\boldsymbol{c} \mid p(\boldsymbol{c}) \in \mathsf{lm}(\mathcal{P}^{\mathcal{I}})\}$.

5.2 The reduction

We proceed by illustrating the tranlation of service specifications into logic programs. Syntactically, such programs are like queries, but have the unrestricted semantics, like our service descriptions; so they provide a nice intermediate step for the complete reduction of service comparison to certain answers. In order to simplify the presentation, we assume that service specifications are normalized by replacing subexpressions with new services,

[1] This notion differs from the many hybrid combinations of rules and DLs (see [7] for a survey). The latter are still rather close to querying DL KBs and their answers are restricted to the constants occurring in the rules or in the KB. Moreover, the purpose is different: those combination are supposed to be knowledge representation formalisms, while our semantics is merely a technical device to link service comparison to query answering against DL KBs.

so that *no constructs are nested* (all subexpressions are service names). We use further service names to guarantee that *if a dataflow graph has more than one functional node, then all nodes are labelled with service names only*. Finally, we assume that message attributes are renamed so that *different functional nodes never share any message attribute name*. Clearly, the above normalizations take polynomial time.

Then for each service name S defined in the specification, we define an atom $p_S(X_{f_1}, \dots, X_{f_m}, Y_{g_1}, \dots, Y_{g_n})$, where p_S is a fresh predicate symbol, and $f_1, \dots, f_m$ (resp. $g_1, \dots, g_n$) is the lexicografic ordering of $\mathsf{in}(S)$ (resp. $\mathsf{out}(S)$). We denote the above atom by H_S.

Now each service S whose dataflow graph has multiple functional nodes with labels $S_1, \dots, S_n$, can be translated into one rule $(H_S \leftarrow H_{S_1}, \dots, H_{S_n})\sigma$, where the substitution σ unifies all variables Y_{g_i} and X_{f_j} such that some parameter node has an incoming edge labelled with g_i and an outgoing edge labelled f_j.

Next consider an S whose dataflow has a single functional node labelled E. If E is $\mathtt{union}(S_1, \dots, S_n)$ then S can be translated into n rules $H_S \leftarrow H_{S_i}$ $(1 \leq i \leq n)$. Symmetrically, if $E = \mathtt{intersection}(S_1, \dots, S_n)$ then S can be translated into one rule $H_S \leftarrow H_{S_1}, \dots, H_{S_n}$.

When $E = \mathtt{if}\, c_1, \dots, c_n\, \mathtt{then}\, S_1\, \mathtt{else}\, S_2$, S is translated into the rules $H_S \leftarrow [c_1], \dots, [c_n], S_1$ and $H_S \leftarrow [\bar{c}_i], S_2$ $(1 \leq i \leq n)$. Here each c_i is a condition and $[c_i]$ denotes its tranlation; $\bar{c}_i$ denotes the complement of c_i, e.g. if c_i is $x = y$ then $\bar{c}_i$ is $x \neq y$; if $c_i = A(x)$ then $\bar{c}_i = \neg A(x)$. The translation $[c_i]$ consists in turning each message attribute f into the corresponding variable X_f.

The translation of $\mathtt{select}\, a_1 := r_1, \dots, a_n := r_n\, \mathtt{from}\, \mathtt{all}\, A\, \mathtt{with}\, p_1 = t_1, \dots p_m = t_m$ is $H_S \leftarrow A(Z), [a_1 := r_1], \dots, [a_n := r_n], [p_1 = t_1], \dots, [p_m = t_m]$, where Z is a fresh variable. Each $[a_i := r_i]$ consists of the translation of the role path r_i into a conjunction of binary atoms (using fresh variables at the intermediate steps), plus the atom $Y_{a_i} = V$, where V is the last variable introduced in the translation of r_i. Similarly, each $[p_i = t_i]$ consists of the translation of the role path p_i plus $u = V$, where V is the last variable introduced in the translation of p_i, and $u = t_i$ if $t_i \in \mathsf{N_c}$, otherwise (i.e. if $t_i \in \mathsf{N_a}$) $u = X_{t_i}$.

Example 1. In our running example, a condition like `hasAddr.hasStr=street` is translated into $\mathtt{hasAddr}(Z, V_1),\ \mathtt{hasStr}(V_1, V_2),\ X_{\mathtt{street}} = V_2$, where V_1, V_2 are new variables.

Due to space limitations we omit the (straightforward) translation of message restructuring and restrictions.

Let us denote the translation of a specification Σ with P_Σ. The above translation is pretty natural and it is not hard to see that it preserves the meaning of the given normal specification under unrestricted query evaluation, as stated by the following theorem.

Theorem 1. *Let Σ be a normalized service specification and let P_Σ be its translation. Let S be the name of a graph $G \in \Sigma$ and $f_1, \dots, f_m$ (resp. $g_1, \dots, g_n$) be the lexicographic ordering of $\mathsf{in}(S)$ (resp. $\mathsf{out}(S)$).*

Then for all models $\mathcal{W}$ of Σ w.r.t. KB, $(t_1, \dots, t_m, u_1, \dots, u_n) \in \mathsf{u\_ans}(p_S, P_\Sigma, \mathcal{W})$ iff for some message pair $(m, m') \in [\![S]\!]^{\mathcal{W}}$, $m(f_i) = t_i$ and $m'(g_j) = u_j$ $(1 \leq i \leq m,\ 1 \leq j \leq n)$.

The above result can be reformulated in terms similar to query containment. For all predicates p_{S_1} and p_{S_2}, let $p_{S_1} \subseteq_{KB,\Sigma} p_{S_2}$ iff for all models $\mathcal{W}$ of Σ w.r.t. KB, $\mathsf{u\_ans}(p_{S_1}, P_\Sigma, \mathcal{W}) \subseteq \mathsf{u\_ans}(p_{S_2}, P_\Sigma, \mathcal{W})$.

Corollary 1. *For all normalized specifications Σ, $S_1 \sqsubseteq_{KB,\Sigma} S_2$ iff $p_{S_1} \subseteq_{KB,\Sigma} p_{S_2}$.*

6 Complexity results

In this section we exploit Theorem 1 and the many recent complexity results on certain query answers against DL knowledge bases to derive a preliminary set of complexity bounds for our service description language.

In order to illustrate decidable cases and complexity sources, we introduce a uniform notation for the fragments of our service description language $\mathcal{SDL}_{full}$:

– $\mathcal{SD}$ restricts the language by forbidding `union`, `else`, negative conditions (such as $r \neq t$ and $\neg A(t)$), and equality within conditions (equality is allowed in the `with` clause of selections);
– superscripts u and e stand for `union` and `else`, respectively; when they are present, the language supports the corresponding constructs;
– similarly, superscripts $=$, $\neq$ and $\neg$ stand for conditions with equalities, disequalities and concept complements, respectively;
– the superscript k imposes that the maximum nesting level of `union` and `else` is bounded by a constant k.

For example, $\mathcal{SD}^{u,\neq}$ stands for the sublanguage of $\mathcal{SDL}_{full}$ supporting `union` and conditions with disequalities, but neither `else` nor negative conditions like $\neg A(t)$. By $\mathcal{SD}^{k,u,\neq}$ we denote a similar language, where the nesting level of `union` is bounded by a constant k.

In this preliminary paper, we adopt the following reduction to obtain a first set of decidability results and complexity upper bounds:

1. Service comparison in Σ is reduced to unrestricted answer containment in P_Σ by Theorem 1; note that P_Σ can be constructed in polynomial time from Σ;
2. unrestricted answer containment is further reduced to unrestricted containment of CQ$^{\neg,\neq}$/UCQ$^{\neg,\neq}$ by *unfolding* P_Σ; unfolding means that whenever an atom B in the body of some rule r unifies with the head of some rule r', then B is replaced with $\mathsf{body}(r')$ (as in SLD resolution); the process is exhaustively repeated; if multiple rules $r_1, \ldots, r_n$ unify with B, then r is replaced with all n possible rewritings; since P_Σ is acyclic (because Σ is), the unfolding process terminates, however it may increase the size of P_Σ exponentially when some predicates are defined by multiple rules;
3. finally, if (the unfolding of) P_Σ is *positive* (i.e., it contains no negations nor any disequality), then unrestricted answer containment in the unfolded version of P_Σ is reduced to certain answering of CQs/UCQs against DL knowledge bases, see Theorem 2 below.

Theorem 2 says that there exists a PTIME reduction of unrestricted CQ (resp. UCQ) containment to the evaluation of certain answers of CQs (resp. UCQs) against DL knowledge bases.

Theorem 2. *Let Σ be a normalized specification and let P^U_Σ be the unfolding of P_Σ. For $i = 1, 2$, let Q_i be the definition of p_{S_i}, i.e. the set of rules $r \in P^U_\Sigma$ with p_{S_i} in* $\mathsf{head}(r)$ *(where S_1 and S_2 are the names of two graphs in Σ).*

If P_Σ is positive, then checking whether $p_{S_1} \subseteq_{KB,\Sigma} p_{S_2}$ can be reduced in polynomial time to evaluating for all $q \in Q_1$ an answer $\mathsf{c\_ans}(Q_2, KB_q)$*, where KB_q is obtained from KB by binding the variables in q to fresh constants, and adding the instantiated body to KB's ABox as a set of assertions.*

More precisely, for each $q \in Q_1$, one has to check whether the tuple of fresh constants assigned to the variables in $\mathsf{head}(q)$ belongs to $\mathsf{c\_ans}(Q_2, KB_q)$. Basically, the reduction is centred around a form of skolemization.

Note that this result is slightly different from the known relationships between query answering and query containment, since $p_{S_1} \subseteq_{KB,\Sigma} p_{S_2}$ is based on a nonstandard (unrestricted) notion of evaluation, similar to the one used for service comparison.

The above reduction suffices to derive complexity bounds for positive P_Σ. Note that P_Σ is positive when `else`, $\neq$, and $\neg$ are not supported, that is, in $\mathcal{SD}^u$ and its fragments. When Σ is formulated in $\mathcal{SD}^u$, then the unfolding of P_Σ may be exponentially larger, as the translation of unions into rules introduces predicates defined by multiple rules. It is not hard to see, however, that $\mathcal{SD}^{k,u}$ specifications lead to unfoldings that are only polynomially larger than P_Σ. Then the above reduction steps tell us that complexity of service comparison within $\mathcal{SD}^{k,u}$ and its fragments is bounded by the complexity of computing certain answers against DL KBs; for $\mathcal{SD}^u$ there is a further exponential explosion due to unfolding.

The complexity of query answering is NP-complete for $\mathcal{EL}$ [9], EXPTIME-complete for $\mathcal{DLR}$ [3], and co3NEXPTIME complete for $\mathcal{SHIQ}$ (cf. [6]). Moreover, query containment w.r.t. empty knowledge bases is NP-hard [4], and it is not difficult to see that the complexity of the standard reasoning tasks in $\mathcal{ALC}$ with general TBoxes (EXPTIME-complete) provides a lower bound to CQ answering against $\mathcal{ALC}$ KBs, so the upper bounds for $\mathcal{EL}$ and $\mathcal{ALC}$ are strict. These observations support the following theorem:

Theorem 3. *The complexity of service comparison in $\mathcal{SD}(\mathcal{X})$ and $\mathcal{SD}^{k,u}(\mathcal{X})$ is*

- *NP-complete for $\mathcal{X} = \mathcal{EL}$;*
- *EXPTIME-complete for $\mathcal{X}$ ranging from $\mathcal{ALC}$ to $\mathcal{DLR}$;*
- *in co3NEXPTIME for $\mathcal{X} = \mathcal{SHIQ}$.*

If the underlying description logic supports unrestricted negation (or equivalently, atomic negation and GCI), then negative literals in rule and query bodies (if any) can be *internalized* in the KB in a simple way: just replace each literal $\neg A(t)$ with $\bar{A}(t)$ where $\bar{A}$ is a fresh atom, and extend the TBox with the axioms $\bar{A} \sqsubseteq \neg A$ and $\neg A \sqsubseteq \bar{A}$. Internalization makes it possible to support constructs such as negated conditions, disequalities, and `else`, that introduces negation implicitly through the translations $[\bar{c}_i]$. After removing negation from P_Σ via internalization, we can exploit the available complexity results for the extensions of $\mathcal{ALC}$ (that allow internalization).

Theorem 4. *The complexity of service comparison in $\mathcal{SD}^\neg(\mathcal{X})$, and between $\mathcal{SD}^{k,e}(\mathcal{X})$ and $\mathcal{SD}^{k,u,e,\neg}(\mathcal{X})$ is*

- *in EXPTIME for $\mathcal{X} = \mathcal{EL}$;*

– *EXPTIME-complete for $\mathcal{X}$ ranging from $\mathcal{ALC}$ to $\mathcal{DLR}$;*
– *in co3NEXPTIME for $\mathcal{X} = \mathcal{SHIQ}$.*

Remark 1. $\mathcal{EL}$ does not support negation, therefore internalization is not possible. In the above theorem we inherit the upper bound for $\mathcal{ALC}$, but whether this is a tight bound is still an open question.

From the above results, we derive upper complexity bounds for more general logics, without any nesting bounds. In the absence of nesting bounds and in the presence of disjunctive constructs like unions and conditionals, the unfolding of P_Σ may be exponential.

Theorem 5. *The complexity of service comparison in $\mathcal{SD}^{u,e,\neg}(\mathcal{X})$ is*

– *in 2-EXPTIME for $\mathcal{X} = \mathcal{EL}$;*
– *in 2-EXPTIME for $\mathcal{X}$ ranging from $\mathcal{ALC}$ to $\mathcal{DLR}$;*
– *in 4-EXPTIME for $\mathcal{X} = \mathcal{SHIQ}$.*

Also in this case, whether these bounds are tight is still an open question.

Currently, we do not know whether $\mathcal{SDL}_{full}$ or even its fragment $\mathcal{SD}^{\neq}$ are decidable. There exist some undecidability results for CQs and UCQs with disequalities, and we conjecture they can be carried over to service comparison. This will be a subject for future work.

	$\mathcal{EL}$	$\mathcal{ALC}$ $\mathcal{DLR}$	$\mathcal{SHIQ}$
$\mathcal{SD}$, $\mathcal{SD}^{k,u}$	NP-complete	EXPTIME-complete	in co3NEXPTIME
$\mathcal{SD}^{\neg}$	(in EXPTIME)	EXPTIME-complete	in co3NEXPTIME
$\mathcal{SD}^{k,e}$, $\mathcal{SD}^{k,e,\neg}$ $\mathcal{SD}^{k,u,e}$, $\mathcal{SD}^{k,u,e,\neg}$	(in EXPTIME)	EXPTIME-complete	in co3NEXPTIME
$\mathcal{SD}^{u}$, $\mathcal{SD}^{e}$, $\mathcal{SD}^{u,e}$ $\mathcal{SD}^{u,\neg}$, $\mathcal{SD}^{e,\neg}$, $\mathcal{SD}^{u,e,\neg}$	(in 2-EXPTIME)	in 2-EXPTIME	in 4-EXPTIME

Table 1. Some complexity results for decidable cases

7 Related work

The language introduced in [1], $\mathcal{SDL}(\mathcal{X})$, was based on an embedding of service comparison into subsumption in an expressive description logic, $\mu\mathcal{ALCIO}$. With the reduction to query containment we adopt here, it is possible to support service intersection and dataflow graphs, even if they violate the quasi-forest structure of $\mu\mathcal{ALCIO}$. Moreover, we provide an articulated complexity analysis not available in [1].

The idea of formalizing services as queries has been first introduced in [5]. The language adopted there is simpler than ours: only one construct combining our selection and restriction, and a form of composition where output and input messages must perfectly match. The semantics of services in [5] is restricted to the constants occurring in a

KB rather than domain elements. Furthermore, all upper bounds provided there are EXPTIME or beyond. Currently our NP bounds for $\mathcal{EL}$ identify the most efficient service description logics in the literature. Moreover, even in the hardest cases, our language is never more complex than [5].

In OWL-S, services are described by means of preconditions, postconditions, and add/delete lists. Pre- and postconditions are like ABoxes; add/delete lists specify the side effects of the services. The same mechanism can describe functional services. WSMO is built upon an articulated model, including user roles and goals, that lead to a planning-like view of service composition. In WSDL-S, WSDL service specifications (that are basically type definitions) are bound to concepts defined in an underlying ontology. No good computational results are currently available for any of the above standards.

8 Conclusions and future work

$\mathcal{SDL}_{full}$ and its fragments are rich service description languages that—however—enjoy numerous decidability results (reported in Table 1), and in some case ($\mathcal{SD}^{k,u}(\mathcal{EL})$) service comparison is significantly less complex than in previous competing logics. Encouraging experimental results are available for an analogous problem [2]. We are planning an experimental implementation based on the same technology.

Many issues need further work, here we mention just the main open problems. Automated service composition needs efficient heuristics for quickly selecting promising candidate dataflows. The bounds for $\mathcal{EL}$ reported in parentheses in Table 1 are simply inherited from more complex logic and it is not obvious whether they are tight. Disequalities and negation over roles would be helpful, but the undecidability results of [8] warn that some restrictions may be needed. It would also be interesting to check whether service comparison can be in NP also for other low-complexity logics such as the *DL-lite* family.

References

1. Piero A. Bonatti. Towards service description logics. In *JELIA, LNCS 2424*, pages 74–85. Springer, 2002.
2. Piero A. Bonatti and F. Mogavero. Comparing rule-based policies. In *9th IEEE Int. Work. on Policies for Distributed Systems and Networks (POLICY 2008)*, pages 11–18, 2008.
3. D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.*, 9(3), 2008.
4. A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. Ninth Annual ACM Symp. on Theory of Computing*, pages 77–90, 1976.
5. D. Hull, E. Zolin, A. Bovykin, I. Horrocks, U. Sattler, and R. Stevens. Deciding semantic matching of stateless services. In *Proc. of AAAI 2006*. AAAI Press, 2006.
6. Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. of Automated Reasoning*, 41(1):61–98, 2008.
7. Riccardo Rosati. Integrating ontologies and rules: Semantic and computational issues. In *Reasoning Web, LNCS 4126*, pages 128–151. Springer, 2006.
8. Riccardo Rosati. The limits of querying ontologies. In *ICDT, LNCS 4353*, pages 164–178. Springer, 2007.
9. Riccardo Rosati. On conjunctive query answering in $\mathcal{EL}$. In *Description Logics*. CEUR-WS.org, 2007.