DIP
Data, Information and Process Integration with Semantic Web Services
FP6 – 507483

Deliverable

D1.3
Framework for hybrid, modularized ontology representation and reasoning for semantics-based services

Boris Motik

June 29, 2004
EXECUTIVE SUMMARY

Based on the deliverable D1.1, this deliverable presents a framework for hybrid reasoning in the Semantic Web. In particular, this deliverable provides a technical platform for integrating disparate logical formalisms into a single unified reasoning formalism. In essence, this framework provides interoperability between OWL-DL and function-free Horn rules. This is very important, since a great number of knowledge representation formalisms, such as F-Logic without function symbols and non-monotonic negation, can be embedded into function-free Horn rules.

This deliverable is structured in two main parts. The first one presents the DL-safe rules as the mechanism for integration of various formalisms in a decidable manner. The second one presents the actual framework which operationalizes hybrid reasoning supporting several different formalisms. Hence, this deliverable serves as a specification for the implementation to be realized in D1.2 and D1.4.
This document presents a framework for hybrid reasoning, with an attempt to integrate different logical formalisms in a coherent logical framework. In particular, it shows how to integrate F-Logic and OWL-DL—two prominent knowledge representation formalisms. The basis for interoperability is provided by the concept of so-called DL-safe rules. Many knowledge representation formalisms can be compiled into DL-safe rules, thus enabling hybrid reasoning.
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Galway | NUIG | Prof. Dr. Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
E-mail: chris.bussler@deri.ie  
Tel: +353 91 512460 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. BankInter,  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mminez@bankinter.es  
Tel: 916234238 |
| Berlecon Research GmbH | Berlecon | Dr. Thorsten Wichmann  
Berlecon Research GmbH,  
Oranienburger Str. 32,  
10117 Berlin, Germany  
E-mail: tw@berlecon.de  
Tel: +49 30 2852960 |
| British Telecommunications Plc. | BT | Dr. Thorsten Wichmann  
Berlecon Research GmbH,  
Oranienburger Str. 32,  
10117 Berlin, Germany  
E-mail: tw@berlecon.de  
Tel: +49 30 2852960 |
| Swiss Federal Institute of Technology, Lausanne | EPFL | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
E-mail: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council | Essex | Mary Rowlands  
Essex County Council,  
P.O. Box 11, County Hall, Duke Street,  
Chelmsford, Essex, CM1 1LX,  
United Kingdom  
E-mail: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik | FZI | Andreas Abecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe  
Germany  
E-mail: abecker@fzi.de  
Tel: +49 721 96540 |
<table>
<thead>
<tr>
<th>Company</th>
<th>Contact Information</th>
</tr>
</thead>
</table>
| Institut für Informatik, Leopold-Franzens Universität Innsbruck | Prof. Dieter Fensel  
Institute of computer science  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
E-mail: dieter.fensel@uibk.ac.at  
Phone: +43 512 507 6486 |
| ILOG SA                                                      | Christian de Sainte Marie  
9 Rue de Verdun, 94253, Gentilly, France  
E-mail: csma@ilog.fr  
Tel: +33 1 49082981 |
| inubit AG                                                    | Torsten Schmale,  
inubit AG,  
Lützowstraße 105-106  
D-10785 Berlin, Germany  
E-mail: ts@inubit.com  
Tel: +49 30726112 0 |
| Intelligent Software Components, S.A.                       | Dr. V. Richard Benjamins, Director R&D  
Intelligent Software Components, S.A.  
Pedro de Valdivia 10  
28006 Madrid, Spain  
E-mail: rbenjamins@isoco.com  
Tel. +34 913 349 797 |
| Net Dynamics Internet Technologies GmbH u. Co KG             | Peter Smolle  
Net Dynamics Internet Technologies GmbH u. Co KG  
Prinz-Eugen-Strasse 68-70  
A-1040 Wien, Austria  
E-mail: peter.smolle@netdynamicstech.com  
Tel.: +43 1 503982615 |
| The Open University                                          | Dr. John Domingue  
Knowledge Media Institute,  
The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK  
E-mail: j.b.domingue@open.ac.uk  
Tel.: +44 1908 655014 |
| SAP AG                                                       | Dr. Elmar Dorner  
SAP Research, CEC Karlsruhe  
SAP AG  
Vincenz-Priessnitz-Str. 1  
76131 Karlsruhe, Germany  
E-mail: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
| Sirma AI Ltd.                                                | Atanas Kiryakov,  
Ontotext Lab, - Sirma AI EAD,  
Office Express IT Centre, 3rd Floor  
135 Tzarigradsko Chausse, Sofia 1784, Bulgaria  
E-mail: atanas.kiryakov@sirma.bg  
Tel.: +359 2 9768 303 |
| Tiscali Osterreich GmbH | Tiscali | Dr Dieter Haacker  
Tiscali Osterreich GmbH.  
Diefenbachgasse 35,  
A-1150 Vienna, Austria  
E-mail: Dieter.Haacker@at.tiscali.com  
Tel: +43 1 899 33 160 |
|------------------------|---------|---------------------------------------------------|
| Unicorn Solution Ltd. | Unicorn | Jeff Eisenberg  
Unicorn Solutions Ltd.  
Malcha Technology Park 1  
Jerusalem 96951,  
Israel  
E-mail: Jeff.Eisenberg@unicorn.com  
Tel.: +972 2 6491111 |
| Vrije Universiteit Brussel | VUB | Carlo Wouters,  
Starlab- VUB  
Vrije Universiteit Brussel  
Pleinlaan 2, G-10  
1050 Brussel, Belgium  
E-mail: carlo.wouters@vub.ac.be  
Tel.: +32 (0) 2 629 3719 |
# Table of Contents

1 Introduction ........................................... 1

2 Preliminaries ........................................... 4
   2.1 Disjunctive Datalog ................................. 4
   2.2 Description Logics ................................. 5
   2.3 Basic Superposition Calculus ..................... 7

3 Integrating Rules and Description Logics ................. 9
   3.1 Reasons for Undecidability of OWL-DL with Rules ..... 9
   3.2 DL-safe Rules ..................................... 11
      3.2.1 Expressivity of DL-safe Rules ............... 11
      3.2.2 Decidability of Query Answering ............. 13

4 Framework for Hybrid Reasoning ......................... 14
   4.1 Framework Overview ................................ 14
   4.2 Reducing DL to Disjunctive Datalog ............... 16
   4.3 Reducing F-Logic to Disjunctive Datalog .......... 18
   4.4 Query Answering in Disjunctive Datalog .......... 20
      4.4.1 Overview ................................... 20
      4.4.2 Formalization ................................ 21

5 Satisfaction of Requirements .......................... 23

6 Conclusion and Future Work .......................... 24
1 Introduction

OWL-DL [35] is a prominent member of the family of languages recommended by W3C for ontology representation in the Semantic Web. It is actually a syntactic variant of the $SHOIN(D)$ description logic, offering a high level of expressivity, while still being decidable. A related language $SHIQ(D)$, distinguished from $SHOIN(D)$ mainly by the absence of nominals, has been successfully implemented in practical reasoning systems, such as Racer [20] or FaCT [22]. Description logics have proven themselves useful in numerous applications, such as information integration [17, 30], software engineering [2, ch. 11] or conceptual modeling [2, ch. 10].

Recently, OWL has been taken as the basis for OWL-S\footnote{http://www.daml.org/services/owl-s/1.0/owl-s.html}, a language for modeling Web services. An important aspect of any service modeling language is modeling service pre- and postconditions. A well-known solution is to represent them using rules [36]. Unfortunately, OWL cannot express arbitrary axioms. We briefly explain the origins of this restriction. Namely, OWL allows using existential quantifiers in the axiom consequents. For example, one may state that “each person has a parent who is a person”. For any person, this axiom implies existence of an infinite sequence of parents. Intuitively, reasoning about an infinite set of objects can lead to undecidability. This intuition is confirmed by a well-known fact that the logic of function-free Horn clauses extended with existential quantifiers is in deed undecidable.

Undecidability is generally considered undesirable in practice. Applications with non-terminating algorithms get stuck in an infinite loop for some inputs, which is typically broken by imposing a resource (usually time) limit. This makes the algorithms lose completeness, so the quality of the results that such applications compute is reduced. Furthermore, even if of high worst-case computational complexity, decidable algorithms are usually more amenable to optimizations. For example, optimization techniques for description logics have been developed, achieving good performance in practice [2, ch. 9].

There are several ways to make query answering in an ontology language decidable. One possibility, taken by disjunctive datalog [14], is to prohibit existential quantifiers. Thus it is possible to reason only about objects explicitly present in the ontology. For finite ontologies, the number of such objects is finite as well, so query answering is decidable. However, this significantly reduces expressivity, since existential quantifiers allow for representing incomplete information. For example, from axioms “Paul has a son” and “a father is someone who has a son”, one may infer that “Paul is a father”, even without knowing the son’s exact identity. Such capabilities are quite useful in practical applications. For example, in a data integration scenario, one might represent missing pieces of information using existential quantifiers. When using existential quantifiers, decidability can be achieved only by restricting the way in which quantifiers are used.

The Web Service Modeling Ontology (WSMO) consortium\footnote{http://www.wsmo.org/} has recently been formed to develop an ontology for modeling semantically-enriched services. An important goal of the consortium is to define the underpinning Web Service Modeling Language (WSML)\footnote{http://www.wsmo.org/wsml/}. The language is still under construction, but (various subsets of) OWL-DL and F-Logic are considered the main candidates. However, it has already...
been noted by many that both of these formalisms do have their benefits and their drawbacks. Hence, instead of choosing a particular formalism, it is more beneficial to reap the benefits of both formalisms.

Obviously, the requirements on ontology languages for the Semantic Web are often contradictory. In this deliverable, we propose a decidable combination of OWL-DL with rules, where decidability is due to restricting the rules to so-called DL-safe ones. Intuitively, the components are not restricted, but only the interface between them is. Generalizing the approach of other decidable combinations of rules and description logics [29, 13], in DL-safe rules, concepts and roles are allowed to occur in both rule bodies and heads as unary respectively binary predicates in atoms, but each variable of a rule is required to occur in some body literal whose predicate is neither a concept nor a role. We discuss the expressive power and the limitations of our approach by means of an example, and show that query answering in such a combination is decidable. In this deliverable we are primarily concerned with the semantic and decidability aspects of hybrid reasoning, and not with the infrastructure aspects, such as the syntax or the exchange of rule definitions on the Web. Concerning these issues, we refer the reader to [23] since our approach is fully compatible with the one proposed there.

DL-safe rules provide a framework for interoperability between different logical formalisms without jeopardizing decidability. Furthermore, DL-safe rules do not require reducing the component formalisms, but reduce the interface between them. In this sense, they fulfill the requirements “5.2: Level of Interoperability” and “5.3: Decidability” of D1.1 [32]. Because of their ability to integrate different logics, DL-safe rules might provide a logical backbone for WSML.

Many knowledge representation formalisms can be represented in the rule-based framework. It is well-known that the LP fragment of F-Logic [28] (without non-monotonic features) can be embedded into the formalisms of Horn clauses [39]. For F-Logic to be decidable, function symbols should not be used in which case F-Logic can be represented using DL-safe rules. Hence, DL-safe rules can be used to enable interoperability between OWL-DL and F-Logic knowledge bases. Thus our framework supports the requirement “5.2: Level of Interoperability” and “5.1: Supported Formalisms” of D1.1 [32].

DL-safe rules do not impose constraints on the rule structure. Arbitrary axioms can automatically be converted into DL-safe rules, without changing the semantics for explicitly named individuals (for details see Chapter 3). Hence, our framework supports the requirement “5.6: Support for Arbitrary Axioms” of D1.1 [32].

To support practical reasoning, we outline a physical framework for hybrid reasoning. The goal of this framework is to provide methodology and algorithms for integrating target formalisms, namely, OWL-DL and F-Logic, with DL-safe rules. One important component of our framework is the reduction of $\mathcal{SHIQ(D)}$ knowledge bases to disjunctive datalog programs [26]. Another component is the translation of the LP fragment of F-Logic into Horn logic. Using these algorithms it is possible to operationalize the logical framework outlined before.

Furthermore, we give a query answering algorithm for DL-safe rules which generalizes the least fixpoint operator, well-known from deductive databases. In such a way, we obtain a query answering algorithm which follows the principle of “graceful degradation”: the user “pays” only for the features she actually uses. Furthermore,
the algorithm is compatible with magic sets transformation [8] and its extension to disjunctive programs [19]. Hence, we believe that our framework can be efficiently realized in practice, and thus fulfill the requirements “5.7: Acceptable Performance Level” and “5.8: Follow the Principle of Graceful Degradation”.

Currently, the algorithms from [26] are not capable of supporting all constructs from OWL-DL. In particular, they do not support nominals; a feature of description logics known to be difficult to handle. Including support for full OWL-DL is the main focus of our future work.

Similarly, our algorithms currently support only monotonic logics. It is well-known that non-monotonic features, such as closed-world assumption or default rules, are difficult to computationally handle in the presence of existential quantifiers. In fact, there is currently no consensus on the semantics of such knowledge bases. Hence, our framework currently does not support requirements “5.4: Open- and Closed-World Assumption” and “5.5: Support for Non-monotonic Negation” of D1.1 [32]. Fulfilling these requirements is another main aspect of our future work. Since disjunctive datalog has been studied extensively as a platform for non-monotonic reasoning, we are quite confident that these extensions will be feasible.

This deliverable focuses primarily on the aspects of hybrid reasoning and its modularization, by integrating various knowledge representation formalisms into a unifying framework. It does not address modularization of ontologies, which is the focus of “D2.2: A framework for representing ontologies consisting of several thousand concept definitions”.

3
2 PRELIMINARIES

2.1 Disjunctive Datalog

In this section we briefly present the syntax and semantics of disjunctive datalog. This presentation is standard and may be found in [14, 19].

A relational schema \( R \) is a finite list of relation symbols \((R_1, \ldots, R_k)\), where each relation is of some arity, written as \( \text{arity}(R_i) \). A relational database \( D \) over \( R \) and a countable domain \( U \) is a finite structure \((U, r_1, \ldots, r_k)\) where \( r_i \) are finite relations over \( U^{\text{arity}(R_i)} \). For a relation symbol \( R_i \), the corresponding relation \( r_i \) from a database \( D \) is often denoted as \( D(R_i) \). \( D \) is sometimes also called an instance of \( R \).

A disjunctive datalog program \( P \) is a triple \((\pi, E, I)\), where \( E \) is a relational schema called extensional schema, \( I \) a relational schema called intensional schema, \( E \) and \( I \) are defined over the same domain \( U \), and \( \pi \) is a finite set of rules of the form

\[
A_1 \lor \cdots \lor A_n \leftarrow B_1, \ldots, B_m
\]

where \( n \geq 0, m \geq 0 \), atoms \( A_i \) and \( B_i \) are of the form \( S(t_1, \ldots, t_n) \) with \( t_i \) being a variable or a constant from \( U \). For atoms \( A_i \), \( S \in I \cup \{\approx\} \), whereas for atoms \( B_i \), \( S \in E \cup I \cup \{\approx\} \). For a rule \( r \), the set of atoms \( \text{head}(r) = \{A_i\} \) is called the rule head, whereas the set of atoms \( \text{body}(r) = \{B_i\} \) is called the rule body. A ground rule with an empty body is called a fact.

Datalog rules are required to be safe, that is, each variable occurring in a head literal must occur in a body literal as well. In this way the explicit reference to the universe of the program is not needed. Typical definitions of disjunctive datalog program, e.g. from [14, 19], allow negated atoms in the body. This negation, however, is non-monotonic, and is different from negation in first-order logic. As our approach produces only positive disjunctive datalog programs, we omit negation from our definitions.

The semantics of disjunctive datalog programs is defined as follows. Let \( P \) be a disjunctive datalog program and let \( D \) be an instance of the extensional schema of \( P \). Then \( P_D = P \cup \{S(t) \mid t \in D(S)\} \) denotes a datalog program obtained by adding to \( P \) each tuple from \( D \) as a fact. The set \( \text{HU}_{P_D} \) is called the Herbrand universe of \( P_D \) and contains all constants from \( P_D \). The ground instance of \( P \) over \( \text{HU}_{P_D} \), written \( \text{ground}(P, \text{HU}_{P_D}) \), is the set of ground rules obtained by replacing variables in each rule of \( P \) with constants from \( \text{HU}_{P_D} \) in all possible ways. The Herbrand base \( \text{HB}_{P_D} \) of \( P_D \) is the set of all ground atoms defined by relations from \( E, I \) and \( \approx \). An interpretation \( M \) of \( P_D \) is a subset of \( \text{HB}_{P_D} \). We say that some ground atom \( A \) is true in an interpretation \( M \) if \( A \in M \). Similarly, \( A \) is false in \( M \) if \( A \notin M \). Interpretation \( M \) is a model of \( P_D \) if, for each rule \( r \in \text{ground}(P, \text{HU}_{P_D}) \), if \( \text{body}(r) \subseteq M \), then \( \text{head}(r) \cap M \neq \emptyset \) and if all atoms from \( M \) involving the \( \approx \) predicate yield an equality relation. An equality relation is a relation that is reflexive, symmetric, transitive, and, for any relation symbol \( R \in E \cup I \), if \( R(\ldots, a, \ldots) \in M \) and \( a \approx b \in M \), then \( R(\ldots, b, \ldots) \in M \) as well.

A model \( M \) is minimal if no subset of \( M \) is a model. The semantics of \( P_D \) is denoted by \( \mathcal{M}_M(P) \) and defined to be the set of all minimal models of \( P_D \). Finally, we define the notion of query answering. A ground literal \( A \) is a cautious answer of \( P \) (written \( P \models_c A \)) if all minimal models of the program contain \( A \); \( A \) is a brave
answer of $P$ (written $P \models_A A$) if at least one minimal model of the program contains $A$. First-order entailment is analogous to cautious entailment.

The size of a rule $r$ is defined as $|r| = 1 + \sum_{1 \leq i \leq n} |A_i| + \sum_{1 \leq j \leq m} |B_j|$, where the size of atoms $A_i$ and $B_j$ is defined as $|S(t_1, \ldots, t_n)| = 1 + n$: predicates and terms are encoded with one symbol, and the leading 1 in the definition of $|r|$ accounts for the implication symbol separating the head from the body. The size of a program $P$, written $|P|$, is the sum of the sizes of all its rules.

2.2 Description Logics

OWL-DL is a syntactic variant of the $\text{SHOIN}(D)$ description logic [23]. Hence, although several XML and RDF syntaxes for OWL-DL exist, in this paper we use the traditional description logic notation, since it is more compact. For the correspondence between this notation and various OWL-DL syntaxes, see [23].

$\text{SHOIN}(D)$ supports reasoning with concrete datatypes, such as strings or integers. For example, it is possible to define a minor as a person whose age is less than or equal to 18 in the following way: $\text{Minor} \equiv \text{Person} \cap \exists \text{age.} \leq 18$. Instead of axiomatizing concrete datatypes in logic, $\text{SHOIN}(D)$ employs an approach similar to [3], where the properties of concrete datatypes are encapsulated in so-called concrete domains. A concrete domain is a pair $(\Delta_D, \Phi_D)$, where $\Delta_D$ is the interpretation domain, and $\Phi_D$ is a set of concrete domain predicates that come with an arity $n$ and a predefined interpretation $d^D \subseteq \Delta^n_D$. An admissible concrete domain $D$ is equipped with a decision procedure for the satisfiability of finite conjunctions over concrete domain predicates. Satisfiability checking of admissible concrete domains can successfully be combined with logical reasoning for many description logics.

We use a set of concept names $N_C$, sets of abstract and concrete individuals $N_{I_a}$ and $N_{I_c}$, respectively, and sets of abstract and concrete roles $N_{R_a}$ and $N_{R_c}$, respectively. An abstract role is an abstract role name or the inverse $S^{-}$ of an abstract role name $S$ (concrete roles do not have inverse roles). In the following, we assume that $D$ is an admissible concrete domain. Intuitively, an abstract role relates two abstract individuals (e.g. an abstract role parent relates a child with his parent), whereas a concrete role relates abstract and concrete individual (e.g. a concrete role age specifies the person’s age).

An $\text{TBox}$ $R$ consists of a finite set of transitivity axioms $\text{Trans}(R)$ and role inclusion axioms of the form $R \subseteq S$ and $T \subseteq U$, where $R$ and $S$ are abstract roles, and $T$ and $U$ are concrete roles. The reflexive-transitive closure of the role inclusion relationship is denoted with $\sqsubseteq^*$. A role not having transitive subroles (w.r.t. $\sqsubseteq^*$, see [24]) is called a simple role.

The set of $\text{SHOIN}(D)$ concepts is defined by the following syntactic rules, where $A$ is an atomic concept, $R$ is an abstract role, $S$ is an abstract simple role, $T(i)$ are concrete roles, $d$ is a concrete domain predicate, $a_i$ and $c_i$ are abstract and concrete individuals, respectively, and $n$ is a non-negative integer:

\[
C \rightarrow A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq n S \mid \leq n S \mid \{a_1, \ldots, a_n\} \mid \geq n T \mid \leq n T \mid \exists T_1, \ldots, T_n.D \mid \forall T_1, \ldots, T_n.D
\]

\[
D \rightarrow d \mid \{c_1, \ldots, c_n\}
\]

A $\text{TBox}$ $T$ consists of a finite set of concept inclusion axioms $C \sqsubseteq D$, where $C$ and $D$ are concepts. An $\text{ABox}$ $A$ consists of a finite set of concept membership
Mapping Concepts to FOL

\[
\begin{align*}
\pi_y(\top, X) &= \top \\
\pi_y(A, X) &= A(X) \\
\pi_y(C \cap D, X) &= \pi_y(C, X) \land \pi_y(D, X) \\
\pi_y(C \cup D, X) &= \pi_y(C, X) \lor \pi_y(D, X) \\
\pi_y(\forall R.C, X) &= \forall y : R(X, y) \rightarrow \pi_x(C, y) \\
\pi_y(\exists R.C, X) &= \exists y : R(X, y) \land \pi_x(C, y) \\
\pi_y(A) &= A(a) \\
\pi_y(b) &= b \\
\pi_y(y, z) &= y(z) \\
\pi_y(x) &= x
\end{align*}
\]

Mapping Axioms to FOL

\[
\begin{align*}
\pi(C(a)) &= \pi_y(C(a)) \\
\pi(R(a, b)) &= R(a, b) \\
\pi(a \approx b) &= a \approx b \\
\pi(a \neq b) &= a \neq b \\
\pi(C \subseteq D) &= \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x) \\
\pi(R \subseteq S) &= \forall x, y : R(x, y) \rightarrow S(x, y) \\
\pi(\text{Trans}(R)) &= \forall x, y, z : R(x, y) \land R(y, z) \rightarrow R(x, z)
\end{align*}
\]

Mapping KB to FOL

\[
\pi(KB) = \bigwedge \forall x, y, z, x : R(x, y) \leftrightarrow R^*(y, x) \land \bigwedge \forall a \in KB_R \cup KB_T \cup KB_A \pi(a)
\]

where \(X\) is a meta variable and is substituted by the actual variable and \(\pi_x\) is defined as \(\pi_y\) by substituting \(x\) and \(x_i\) for all \(y\) and \(y_i\), respectively.

Table 2.1: Translation of SHOIN(D) into FOL

axioms \(C(a)\), role membership axioms \(R(a, b)\) and \(T(a, b^c)\), and individual equalities and inequalities \(a^{(c)} \approx b^{(c)}\) and \(a^{(c)} \neq b^{(c)}\), where \(C\) is a concept, \(R\) an abstract role, \(T\) a concrete role, \(a\) and \(b\) abstract individuals, and \(a^c\) and \(b^c\) concrete individuals. A SHOIN(D) knowledge base \((T, R, A)\) consists of a TBox \(T\), an RBox \(R\), and an ABox \(A\).

The SHIQ(D) description logic is obtained from SHOIN(D) by disallowing nominal concepts of the form \(\{a_1, \ldots, a_n\}\) and \(\{c_1, \ldots, c_n\}\), and by allowing qualified number restrictions of the form \(\geq n \cdot S.C\) and \(\leq n \cdot S.C\), for \(C\) a SHIQ(D) concept and \(S\) a simple role.

Since our algorithms are based on resolution, instead of giving a direct model-theoretic semantics to SHOIN(D) [24], we give an equivalent semantics by translation into multi-sorted first order logic \footnote{In multi-sorted first-order logic, variables, constants and function symbols are divided into sorts to separate their interpretation, see [25] for details.}. To separate the interpretations of the abstract and the concrete domain, we introduce the sorts \(a\) and \(c\), and use the notation \(x^c\) and \(f^c\) to denote that \(x\) and \(f\) are of sort \(c\). We translate each atomic concept into a unary predicate of sort \(a\), each \(n\)-ary concrete domain predicate into a predicate with arguments of sort \(c\), and each abstract (concrete) role into a binary predicate of sort \(a \times a\) (\(a \times c\)). The translation operator \(\pi\) is presented in Table 2.1. Intuitively, in \(\pi_y(C, X)\), \(X\) represents the individual or the variable for which the membership in the concept expression \(C\) is stated, and \(y\) determines the names of variables to be used for quantification.
2.3 Basic Superposition Calculus

In some algorithms we present later we make use of the basic superposition calculus, which we briefly overview here. This calculus has been developed to optimize theorem proving with equality [7]. A similar calculus was developed by Nieuwenhuis and Rubio [33].

We assume a standard notion of first-order clauses with equality: all existential quantifiers have been eliminated using Skolemization; all remaining variables are universally quantified; we only consider the equality predicate (all non-equational literals $A$ are encoded as $A \approx \top$ in a multi-sorted setting); and we treat $\approx$ as having built-in symmetry. Moreover, we assume the reader to be familiar with standard first-order resolution [6].

Basic superposition is an optimized version of superposition (a calculus for equational theories [5]) which prohibits superposition into terms introduced by previous unification steps, thus reducing the number of clauses generated. Its inference rules are formalized by distinguishing two parts of a clause: (i) the skeleton clause $C$ and (ii) the substitution $\sigma$ representing the cumulative effects of previous unifications. Such a representation of $C\sigma$ is called a closure, and is written as $C \cdot \sigma$. A closure can conveniently be represented by marking the terms in $C\sigma$ occurring at variable positions of $C$. Any position at or beneath a marked position is called a substitution position. E.g., the clause $P(f(y)) \lor g(b) \approx b$ is logically equivalent to the closure $(P(x) \lor z \approx b) \cdot \{x \mapsto f(y), z \mapsto g(b)\}$, which can conveniently be represented as $P([f(y)]) \lor [g(b)] \approx b$.

The calculus requires two parameters. The first is an admissible ordering on terms $\succ$, i.e., a reduction ordering total on ground terms. Such an ordering is then extended to literals [7]. The second parameter of the calculus is a selection function which selects an arbitrary set of negative literals in a closure.

$L \cdot \sigma$ is maximal in $C \cdot \sigma$ if there is no $L' \in C \setminus \{L\}$ such that $L' \sigma \succ L \sigma$. $L \cdot \sigma$ is strictly maximal in $C \cdot \sigma$ if there is no $L' \in C \setminus \{L\}$ such that $L' \sigma \succeq L \sigma$. A literal $L \cdot \theta$ is (strictly) eligible for superposition (S)ES in a closure $(C \lor L) \cdot \theta$ if nothing is selected in $(C \lor L) \cdot \theta$ and $L \cdot \theta$ is (strictly) maximal in $C \cdot \theta$. A literal $L \cdot \theta$ is eligible for resolution (ER) in a closure $(C \lor L) \cdot \theta$ if it is selected in $(C \lor L) \cdot \theta$ or nothing is selected in $(C \lor L) \cdot \theta$ and $L \cdot \theta$ is maximal in $C \cdot \theta$. With these definitions in mind, the basic superposition calculus, BS for short, consists of the inference rules given below.

Positive superposition:

\[
\frac{(C \lor s \approx t) \cdot \rho (D \lor w \approx v) \cdot \rho}{(C \lor D \lor w[t]_p \approx v) \cdot \theta}
\]

where (i) $\sigma = \text{MGU}(sp, wp)_p$, (ii) $\theta = \rho \sigma$, (iii) $t \theta \not\approx s \theta$ and $v \theta \not\approx w \theta$, (iv) $(s \approx t) \cdot \theta$ is SES, (v) $(w \approx v) \cdot \theta$ is SES, (vi) $(s \theta \approx t \theta \not\approx w \theta \approx v \theta)$, (vii) $w \mid_p$ is not a variable.

Negative superposition:

\[
\frac{(C \lor s \approx t) \cdot \rho (D \lor w \not\approx v) \cdot \rho}{(C \lor D \lor w[t]_p \not\approx v) \cdot \theta}
\]

where (i) $\sigma = \text{MGU}(sp, wp)_p$, (ii) $\theta = \rho \sigma$, (iii) $t \theta \not\approx s \theta$ and $v \theta \not\approx w \theta$, (iv) $(s \approx t) \cdot \theta$ is SES, (v) $(w \not\approx v) \cdot \theta$ is ER, (vi) $w \mid_p$ is not a variable.
Reflexivity resolution: \[
\frac{(C \lor s \not= t) \cdot \rho}{C \cdot \theta}
\]
where (i) $\sigma = \text{MGU}(s\rho, t\rho)$, (ii) $\theta = \rho\sigma$, (iii) $(s \not= t) \cdot \theta$ is ER.

Equality factoring: \[
\frac{(C \lor s \approx t \lor s' \approx t') \cdot \rho}{(C \lor t \not= t' \lor s' \approx t') \cdot \theta}
\]
where (i) $\sigma = \text{MGU}(s\rho, s'\rho)$, (ii) $\theta = \rho\sigma$, (iii) $t\theta \not= s\theta$ and $t'\theta \not= s'\theta$, (iv) $(s \approx t) \cdot \theta$ is ES.

Ordered resolution: \[
\frac{(C \lor A) \cdot \rho (D \lor A) \cdot \rho}{(C \lor D) \cdot \theta}
\]
where (i) $\sigma = \text{MGU}(A\rho, B\rho)$, (ii) $\theta = \rho\sigma$, (iii) $A \cdot \theta$ is SES, (iv) $B \cdot \theta$ is ER.

A derivation from a closure set $N_0$ is a sequence of closure sets $N_0, N_1, \ldots, N_i$, where $N_i = N_{i-1} \cup \{C\}$, and $C$ is derived by applying a BS inference rule with premises from $N_{i-1}$. Roughly speaking, the set of closures $N_i$ is saturated up to redundancy if all inferences from premises in $N_i$ are redundant in $N_i$. If this is the case, then $N_i$ contains the empty closure if and only if $N_0$ is unsatisfiable, so BS is a sound and complete refutation procedure [7]. Compatible redundancy elimination rules have been presented in [7, 25].
3 Integrating Rules and Description Logics

OWL-DL is a standard for ontology modeling in the Semantic Web. Although it offers some frame-like syntactic sugar, OWL-DL is in its essence the SHOIN(D) description logic. As pointed out in Chapter 1, description logics are a decidable formalism. However, decidability is achieved by reducing the expressivity of the logic. These restrictions make OWL-DL unsuitable for certain modeling tasks.

Apart from description logic, numerous other knowledge representation formalisms exist. F-Logic is one such formalism, providing for a modeling style inspired by object-oriented systems. A detailed comparison between description logic and F-Logic has been given in [32]. As noted there, instead of selecting an existing formalism and then living with its good and bad sides, in the context of modern applications it makes sense to consider integrating various formalisms. Proving a platform for integration is the main focus of this deliverable.

As discussed in [32], Horn rules are the logical framework underlying most knowledge representation formalisms. Formalisms such as F-Logic can be easily transformed and executed using a rule-based framework [39]. Hence, it is intuitively clear that the platform for integration should somehow support Horn rules.

It is well-known that adding arbitrary Horn rules to description logic makes the logic undecidable [29]. As outlined in [32], this is an undesirable situation, since it is well-known that decidable logics are more amenable for implementation and optimization in practice. Hence, in this chapter we focus on providing a decidable combination of function-free Horn rules and description logics. This combination serves as the basis of our hybrid reasoning framework. In the following we first discuss why adding rules to description logics does lead to undecidability. Based on these considerations, we define the class of DL-safe rules. We show that adding DL-safe rules to description logics does not lead to undecidability. Hence, we view DL-safe rules as the foundation for a framework for hybrid reasoning, integrating existing formalisms such as OWL-DL and F-Logic.

3.1 Reasons for Undecidability of OWL-DL with Rules

In [23], the following problem was shown to be undecidable: given an OWL-DL knowledge base $KB$ and a datalog program $P$, is there a common model of $KB$ and $P$, i.e. is $KB$ consistent with $P$? As a consequence, subsumption and query answering w.r.t. knowledge bases and programs are also undecidable. Investigating this proof and the ones in [29] more closely, we note that the undecidability is caused by the interaction between some very basic features of description logics and rules. In this section, we try to give an intuitive explanation of this result and its consequences.

Consider the simple knowledge base $KB$ from Table 3.1. It is not too difficult to see that this knowledge base implies the existence of an infinite chain of fathers: since Peter must have a father, there is some $x_1$ who is a Person. In turn, $x_1$ must have some father $x_2$, which must be a Person, and so on. An infinite model with such a chain is shown in Figure 3.1, upper part a). Observe that Peter is a grandchild, since he has a father who is a father.

Let us now check whether $KB \models Grandchild(Jane)$; this is the case if and only if $KB \cup \{\neg Grandchild(Jane)\}$ is unsatisfiable, i.e. if it does not have a model. We can do this by trying to build such a model; if we fail, then we conclude that $KB \cup$
Table 3.1: Example Knowledge Base

\{\neg \text{Grandchild}(Jane)\} \text{ is unsatisfiable. However, we have a problem: starting from } Peter, \text{ a naïve approach to building a model will expand the chain of Peter’s fathers indefinitely, and will therefore not terminate.}

This very simple example intuitively shows that we have to be careful if we want to ensure termination of a satisfiability checking algorithm. For many DLs, termination can be ensured without losing correctness because we can restrict our attention to certain “nice” models:\footnote{To be precise, for some DLs, we can only restrict our attention to “nice abstractions” of models.} for numerous DLs, we can restrict our attention to tree models, i.e. to models where the underlying relational structure forms a tree [38]. This is so because every satisfiable knowledge base has such a tree model. Even if such a tree model is infinite, we can wind this infinite tree model into a finite one. In our example, since KB does not require each father in the chain to be distinct, the model in Figure 3.1, lower part b) is the result of this “winding” of a tree into a “nice” model. Due to their regular structure, these “windings” of tree models are easily constructed in an automated way. To understand why every satisfiable \text{SHIQ(D)} knowledge base has a tree model [24], consider the mapping \pi in Table 2.1 more closely (we abstract some technicalities caused by the transitive roles): all formulae speak about variables related to each other only in some tree-like manner, as underlined by the following example:

\[
\exists S. (\exists R.C \cap \exists R.D) \subseteq Q \\
\forall x : \{[\exists y : S(x,y) \land (\exists x : R(y,x) \land C(x)) \land (\exists x : R(y,x) \land D(x))] \rightarrow Q(x)\} \\
\forall x, x_1, x_2, x_3 : \{S(x,x_1) \land R(x_1,x_2) \land C(x_2) \land R(x_2,x_3) \land D(x_3) \rightarrow Q(x)\}
\]

Let us compare these observations with the reasoning required for function-free Horn rules. In these rules, all variables are quantified universally, i.e. there are no existentially quantified variables in rule consequents. Hence we never have to infer the existence of “new” objects. Thus, reasoning algorithms must consider only individuals explicitly introduced in the knowledge base and will never run into the termination problems outlined above. Hence, the rules are allowed to enforce arbitrary but finite, non-tree relational models, and not only “nice” models.

\begin{center}
\begin{tabular}{c|c}
\hline
\text{Person}(Peter) & Peter is a person. \\
Person \sqsubseteq \exists father.Person & Each person has a father who is a person. \\
\exists father.(\exists father.Person) \sqsubseteq Grandchild & Things having a father of a father who is a person are grandchildren. \\
\hline
\end{tabular}
\end{center}
Now let us see what happens if we combine both. If we extend, e.g. $\text{SHIQ}(\mathbf{D})$ with function-free Horn rules, then we combine a logic whose decidability is due to the fact that we can restrict our attention to “nice” models (but with individuals whose existence may be implied by a knowledge base) with one whose decidability is due to the fact that we can restrict our attention to “known” individuals (but with arbitrary relations between them). Unsurprisingly, this and similar combinations are undecidable [29, 23].

3.2 DL-safe Rules

As a reaction to the observations in Section 3.1, we define DL-safe rules. Then, we discuss the benefits and drawbacks of DL-safe rules and prove that query answering in $\text{SHOIN}$ with DL-safe rules is decidable.

Definition 3.2.1 (DL-safe Rules). Let $KB$ be a $\text{SHOIN}(\mathbf{D})$ knowledge base, and let $N_P$ be a set of predicate symbols such that $N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$. A DL-atom is an atom of the form $A(s)$, where $A$ is a concept name in $KB$, or of the form $R(s,t)$, where $R$ is a role in $KB$. A rule $r$ defined over predicates from $N_P$ is called DL-safe if each variable in $r$ occurs in a non-DL-atom in the rule body. A program $P$ is DL-safe if all its rules are DL-safe.

The semantics of the combined knowledge base $(KB, P)$ is given by translation into first-order logic as $\pi(KB) \cup P$. The main inference in $(KB, P)$ is query answering, i.e. deciding whether $\pi(KB) \cup P \models \alpha$ for a ground atom $\alpha$.

Some remarks are in order. Firstly, DL-safety is similar to the safety in datalog. In a safe rule, each variable occurs in a positive atom in the body, and may therefore be bound only to constants explicitly present in the database. Similarly, DL-safety makes sure that each variable is bound only to individuals explicitly introduced in the ABox. For example, if $\text{Person}$, $\text{livesAt}$, and $\text{worksAt}$ are concepts and roles from $KB$, the following rule is not DL-safe:

$$\text{Homeworker}(x) \leftarrow \text{Person}(x), \text{livesAt}(x,y), \text{worksAt}(x,y)$$

The reason for this is that both variables $x$ and $y$ occur in DL-atoms, but do not occur in an atom with a predicate outside of $KB$. This rule can be made DL-safe by adding literals $O(x)$ and $O(y)$ to the rule, and by adding a fact $O(a)$ for each individual $a$ occurring in the knowledge base. In Subsection 3.2.1 we discuss the consequences that this transformation has on the semantics of the rule.

Secondly, DL-safety only allows atomic concepts to occur in a rule. This is not really a restriction: for a complex concept $C$, one may introduce an atomic concept $A_C$, add the TBox axiom $C \sqsubseteq A_C$ and use $A_C$ in the rule [34].

3.2.1 Expressivity of DL-safe Rules

In our approach, to achieve decidability, we do not restrict the component languages. Rather, we combine full $\text{SHOIN}(\mathbf{D})$ with function-free Horn rules, and thus extend both formalisms. The DL-safety requirement only restricts the interchange of consequences between the components to only those consequences involving individuals explicitly introduced in the ABox.
Fatherhood is a kind of parenthood.
A bad child is a grandchild who hates one of his siblings.
DN-safe version of a bad child.

Table 3.2: Example with DL-safe Rules

<table>
<thead>
<tr>
<th>father ⊆ parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadChild(x) ← Grandchild(x), parent(x, y), parent¬(y, z), hates(x, z)</td>
</tr>
<tr>
<td>BadChild'(x) ← Grandchild(x), parent(x, y), parent¬(y, z), hates(x, z), O(x), O(y), O(z)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person(Cain)</th>
</tr>
</thead>
<tbody>
<tr>
<td>father(Cain, Adam)</td>
</tr>
<tr>
<td>hates(Cain, Abel)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person(Romulus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>∃father.∃father¬.{Remus}(Romulus)</td>
</tr>
<tr>
<td>hates(Romulus, Remus)</td>
</tr>
</tbody>
</table>

| Child(x) ← GoodChild(x), O(x) |
| Child(x) ← BadChild'(x), O(x) |

| (GoodChild ⊆ BadChild')(Oedipus) |

Table 3.2: Example with DL-safe Rules

To illustrate the expressive power of DL-safe rules, we extend the example from Table 3.1 with axioms and rules from Table 3.2. We define a notion of a BadChild as a grandchild which hates some of its siblings (or itself). Notice that this rule involves relations forming a triangle between two siblings and a parent and thus cannot be expressed in a description logic such as SHOIN(D). Moreover, it is not DL-safe because all variables in the rule do not occur in non-DL-atoms the rule body.

Now consider the first group of ABox facts. Since Cain is a Person, as in Section 3.1 one may infer that Cain is a Grandchild. Since Cain and Abel are children of Adam, and Cain hates Abel, Cain is a BadChild.

Similarly, Romulus has a father who is a father of Remus, and Romulus hates Remus, so Romulus is a BadChild as well. We are able to derive this without knowing exactly who the father of Romulus is. Historically, the identity of the father of Romulus and Remus is unknown, but his existence is certain, which is exactly what the knowledge base represents.

Consider now the DL-safe rule defining BadChild' (assuming that the ABox contains O(a) for each individual a in the ABox): since the father of Cain and Abel is known by name (i.e. Adam is in the ABox), this rule implies that Cain is a BadChild'. In contrast, the father of Romulus and Remus is not known in the ABox. Hence, the literal O(y) from the DL-safe rule cannot be matched to the father’s name, so the rule does not derive that Romulus is a BadChild'.

This may seem confusing. However, DL-safe rules do have a ‘natural’ reading: just append the phrase ‘where the identity of all objects is known’ to the meaning of the rule. For example, the rule defining BadChild' can be read as ‘A BadChild' is a known grandchild for which we know a parent, and who hates one of his known siblings’.

Please note that combining description logics with DL-safe rules increases the expressivity of both components. Namely, a SHOIN(D) knowledge base cannot imply that Cain is a BadChild' because the ‘triangle’ rule cannot be expressed in SHOIN(D). Similarly, a set of function-free Horn rules cannot imply this either: we
know that Cain has a grandfather because Cain is a person, but we do not know who he is. Hence we need the existential quantifier to infer the existence of ancestors, and then infer that Cain is a Grandchild.

Finally, we would like to point out that it is incorrect to compute all consequences of the description logic component first, and then to apply the rules to the consequences. Consider the KB part about Oedipus: he is a GoodChild or a BadChild', but we do not know exactly which. Either way, one of the rules derives that Oedipus is a Child, so (KB, P) \models Child(Oedipus). This would not be derived by applying the rules to the consequences of KB, since KB \not\models GoodChild(Oedipus) and KB \not\models BadChild'(Oedipus).

3.2.2 Decidability of Query Answering

We now sketch a proof of the decidability of query answering for the combination of SHOIN knowledge bases with DL-safe rules by a non-deterministic reduction of the query answering problem to the satisfiability problem for SHOIN knowledge bases without rules.

**Theorem 3.2.2.** For a SHOIN knowledge base KB and a DL-safe program P, query answering in (KB, P) is decidable.

**Proof.** Clearly (KB, P) \models \alpha if \pi(KB) \cup P' is unsatisfiable, where P' = P \cup \{\neg \alpha\}. Let P^g be the set of ground instances of P', i.e. P^g contains all possible ground instantiations of rules in P' with individuals from KB. Since each variable occurs in each rule body in a non-DL-atom, we have that \pi(KB) \cup P' is satisfiable iff \pi(KB) \cup P^g is satisfiable.

Satisfiability of \pi(KB) \cup P^g can be decided by case analysis as follows: each model of P^g satisfies at least one literal per rule. Hence, we don’t-know non-deterministically choose one literal per clause in P^g and, for the resulting set of literals L^e, we test the satisfiability of \pi(KB) \cup L^e. Clearly, \pi(KB) \cup P^g is satisfiable iff there exists a ‘choice’ of L^e such that \pi(KB) \cup L^e is satisfiable.

Next, let L_{DL}^e \subseteq L^e be the set of (ground) literals in L^e involving DL predicates. Clearly, \pi(KB) \cup L^e is unsatisfiable iff either L^e contains a complementary pair of ground literals or \pi(KB) \cup L_{DL}^e is unsatisfiable. The first case can be checked easily, and the second case can be reduced to standard SHOIN reasoning as follows: L_{DL}^e can be viewed as an ABox, apart from literals of the form \neg R(a, b). However, each such literal can be transformed into an equivalent SHOIN ABox assertion (\forall R.\neg \{b\})(a). Thus we have reduced query answering to deciding satisfiability of a SHOIN knowledge base. This problem is decidable because (i) transitivity axioms can be eliminated from SHOIN knowledge bases in the same way as this is done for SHIQ in [25] and (ii) the resulting logic is a syntactic variant of the two variable fragment of first order logic with counting quantifiers, which is known to be decidable [18].

We strongly believe that Theorem 3.2.2 also holds for SHOIN(D): (i) the decidability proof of SHOIN should be easily adaptable to SHOIN(D), and (ii) the same non-deterministic reduction of ground DL-safe rules to sets of ground literals is applicable to SHOIN(D). To work out the details of this proof is our future work.
4 Framework for Hybrid Reasoning

In Chapter 3 we have introduced the formalism of DL-safe rules. This formalism provides the foundation for interoperability between description logics and Horn rules, thus providing the foundation for a hybrid reasoning framework, enabling decidable reasoning with both description logics and rules.

So far, we have just shown that this formalism is decidable, but have not discussed how to operationalize it. In this chapter we focus on operationalizing the formalism.

4.1 Framework Overview

The structure of the framework is schematically represented in Figure 4.1. The components addressed in greater detail in this deliverable and our previous work are shown in gray, whereas the components for which more additional work is needed are shown in white. We next describe each component in more detail.

Disjunctive Datalog. To operationalize hybrid reasoning, we choose disjunctive datalog with equality as the main formalism. Disjunctive datalog has numerous benefits, making it suitable for this task:

- It is a rule-based formalism. Rule-based systems, such as Prolog, have often been found suitable for implementing complex reasoning systems. Many target formalisms can directly be embedded into disjunctive datalog. For example, DL-safe rules are themselves datalog rules, so integrating them is straightforward. Similarly, F-Logic can be translated into a rule-based formalism.

- Disjunctive datalog provides so-called “reasoning by cases”, i.e. it supports disjunction in the head of the rule. This feature is crucial, since description logics themselves support reasoning by cases.

- Several optimization strategies are available for disjunctive datalog. In particular, the magic sets transformation [8, 19] has been shown to be very successful in optimizing query answering.

- Disjunctive datalog is a relational formalism, matching closely with the formalism of relational databases. Therefore, integrating existing data sources into the reasoning process is going to be relatively easy [10].

Disjunctive datalog is operationalized in the framework by a disjunctive datalog reasoning engine. Different knowledge representation formalisms are incorporated into the framework by a suitable adaptor component. Certain adaptors are shown in the upper part of Figure 4.1.

Query Answering. Cautious query answering in disjunctive datalog is of higher data complexity than in ordinary datalog [11]: $\Pi_2^P$-complete vs. P-complete. Therefore, a query answering algorithm is needed which will not introduce unnecessary performance penalty. We developed such an algorithm, and present it in Section 4.4. It extends the standard least fixpoint operator [1] in two ways: (i) it supports rules
with disjunctions in the head and \((ii)\) it uses an optimized calculus for handling equality. If disjunction and equality are not used, the algorithm reduces to the least fixpoint operator. In this way, our framework supports the principle of “graceful degradation”: the user pays a performance penalty only if he actually uses an “expensive” primitive.

**OWL-DL.** The framework supports OWL-DL by translating it into disjunctive datalog, as explained in Section 4.2. Actually, this translation currently supports \(SHIQ(D)\) description logic, which does not quite match OWL-DL: it does not support nominals, but does allow qualified number restrictions. It is worth mentioning that current state-of-the-art DL reasoners, such as Racer [20] or FaCT [22] also do not support nominals. Extending the translation to all of OWL-DL will be the main focus of our future work.

**DL-safe Rules.** DL-safe rules are datalog rules themselves, so they can be incorporated into the framework in a straightforward manner. As shown in Section 3.2, both transitive and intransitive roles are allowed to occur in DL-atoms. In our framework, only simple roles (i.e. roles without transitive subroles) are allowed to occur in rules. The reason for this is the transformation we use to handle transitivity axioms, as explained in Section 4.2. Supporting arbitrary roles in DL-safe rules is the part of our future work.

**F-Logic.** The LP variant of F-Logic can be translated into Horn logic, and, if function symbols are not used in the F-Logic knowledge base, then even datalog is sufficient. Hence, our framework supports F-Logic without functional terms, and, once it is extended with Horn rules with function symbols, it will support the Horn fragment of F-Logic as well.
Rules with Function Symbols. Supporting Horn rules with function symbols is not our current focus. However, it is well-known that, if Horn rules are not recursive, query answering is decidable. Hence, we conjecture that extending our query answering algorithm to handle non-recursive Horn rules will be straightforward. This is one of the aspects of our future work.

Non-monotonic Reasoning. Both OWL-DL and datalog are monotonic reasoning formalisms. As such, they do not provide useful non-monotonic constructs, such as default or closed-world reasoning. However, disjunctive datalog has been extensively studied as a platform for non-monotonic reasoning, by allowing non-monotonic negation in rule bodies, interpreted under stable or perfect model semantics [14]. Hence, although our framework does not support non-monotonic reasoning yet, we believe that it can be extended in this direction with moderate effort. To do so, mainly the query answering algorithm will need to be adapted, as it currently supports positive programs only.

API. The applications interface with the framework through an API providing access to each of the components languages. It might be possible to split the API into several subcomponents, each corresponding to one component language. We decided against this practice, since the main challenge for this API is to provide an integrated access to all considered subcomponents.

Data Sources. The datalog engine with equality provides reasoning services, but it does not itself host the data it reasons over. The data reasoned with is provided by different data sources which are plugged into the engine by means of suitable wrappers. Initially, our main focus will be on providing a wrapper for relational databases. In such a way it will be possible to reason over existing legacy data using our framework. The integration with existing data sources is going to be coordinated with WP2, particularly with D2.2 and D2.6.

In the following sections we focus in more detail on certain important components of the framework.

4.2 Reducing DL to Disjunctive Datalog

In this section we outline an algorithm for reducing description logic knowledge bases to disjunctive datalog. Unfortunately, handling all of OWL-DL is theoretically difficult. The main problem are the nominals. Namely, the combination of nominals, inverse roles, and number restriction is known to be difficult to handle, which is confirmed by the increase in complexity from ExpTime to NExpTime [37]. Hence, we focus on SHIQ(D) description logic, which differs from OWL-DL mainly by not supporting nominals. All existing practical systems support currently only this logical fragment.

Furthermore, DL-safe rules, where DL-atoms are restricted to concepts and simple roles, can simply be appended to the result of the transformation. Hence, the positive disjunctive program obtained by our algorithms entails the same set of ground facts as the original knowledge base. For unary coding of numbers and by assuming a bound on the arity of predicates in rules, our algorithm runs in deterministic exponential time, which makes it optimal since SHIQ is ExpTime-complete [37].
The full presentation of the algorithm and a proof of its correctness are technically involved and lengthy. Here, we just provide an overview of the procedure, without going into details. For a complete presentation of the procedure and for the proofs of its correctness, we direct the interested reader to [25, 26].

Let $KB$ be a $SHIQ(D)$ knowledge base. The reduction of $KB$ to a disjunctive datalog program $DD(KB)$ can be computed by an algorithm schematically presented in Figure 4.2. We next explain each step of the algorithm.

Elimination of Transitivity Axioms. Our core algorithms cannot handle transitivity axioms, basically because in their first order logic formulation they involve three variables, which are known to be difficult to handle [27]. However, we can eliminate transitivity axioms by encoding $KB$ into an equisatisfiable knowledge base $\Omega(KB)$. Roughly speaking, for each transitive role $S$, each role $S \sqsubseteq^* R$, and each concept $C$ occurring in $KB$, it is sufficient to add an axiom $\forall R.C \sqsubseteq \forall S.(\forall S.C)$. Intuitively, this axiom propagates all concept constraints through transitive roles. Whereas $KB$ and $\Omega(KB)$ entail the same set of ground facts concerning simple roles, they do not entail the same set of ground facts concerning complex roles. This is the reason for allowing only simple roles to occur in DL-safe rules (see Section 4.1).

Translation into Clauses. The next step is to translate $\Omega(KB)$ into clausal first-order logic. We first use $\pi$ as defined in Table 2.1 and then transform the result $\pi(\Omega(KB))$ into clausal form using structural transformation to avoid an exponential blow-up [34]. We call the result $\Xi(KB)$.

Saturation by Basic Superposition. We next saturate the RBox and TBox of $\Xi(KB)$ by basic superposition [7] – a clausal calculus optimized for theorem proving with equality. In this key step of the reduction, we compute all non-ground consequences of $KB$. We can prove that saturation terminates because application of each rule of basic superposition produces a clause with at most one variable and with functional terms of depth at most two. Indeed, this yields an exponential bound on the number of clauses we may compute, and thus an exponential time complexity bound for our algorithm so far.

Elimination of Function Symbols. Saturation of RBox and TBox of $\Xi(KB)$ computes all non-ground consequences of $KB$. If we add ABox assertions to this saturated clause set, all inferences by basic superposition will produce only ground clauses. Moreover, the resulting ground clauses contain only ground functional terms of depth one. Hence, it is possible to simulate each functional term $f(a)$ with a new constant $a_f$. For each function symbol $f$, we introduce a binary predicate $S_f$, and for each individual $a$, we add an assertion $S_f(a, a_f)$. Finally, if a clause contains the term $f(x)$, we replace it with a new variable $x_f$ and add the literal $\neg S_f(x, x_f)$, as in the following example:

![Figure 4.2: Algorithm for Reducing $SHIQ(D)$ to Datalog Programs](image.png)
\[
\neg C(x) \lor D(f(x)) \Rightarrow \neg S_f(x, x_f) \lor \neg C(x) \lor D(x_f)
\]

We denote the resulting function-free set of clauses with FF(\(KB\)). In [25], we show that each inference step of basic superposition in \(\Xi(\text{KB})\) can be simulated by an inference step in FF(\(\text{KB}\)), and vice versa. Hence, \(\text{KB}\) and FF(\(\text{KB}\)) are equisatisfiable.

**Conversion to Disjunctive Datalog.** Since FF(\(\text{KB}\)) does not contain functional terms and all its clauses are safe, we can rewrite each clause into a positive disjunctive rule. We use \(\text{DD}(\text{KB})\) for the result of this rewriting.

The following theorem summarizes the properties of our algorithm (we use \(\models_c\) for cautious entailment in disjunctive datalog, which coincides with first-order entailment for positive datalog programs [14]):

**Theorem 4.2.1 ([25]).** Let \(\text{KB}\) be an \(\text{SHIQ}(\text{D})\) knowledge base, defined over an admissible concrete domain \(\text{D}\), such that satisfiability of finite conjunctions over \(\Phi_\text{D}\) can be decided in deterministic exponential time. Then the following claims hold:

1. \(\text{KB}\) is unsatisfiable if and only if \(\text{DD}(\text{KB})\) is unsatisfiable.
2. \(\text{KB} \models \alpha\) if and only if \(\text{DD}(\text{KB}) \models_c \alpha\), for \(\alpha\) of the form \(A(a)\) or \(S(a, b)\), \(A\) an atomic concept, and \(S\) a simple role.
3. \(\text{KB} \models C(a)\) for a non-atomic concept \(C\) if and only if, for \(Q\) a new atomic concept, \(\text{DD}(\text{KB} \cup \{C \sqsubseteq Q\}) \models_c Q(a)\).
4. Let \(|\text{KB}|\) be the size of the knowledge base \(\text{KB}\) encoded in the standard way [25], with numbers coded in unary. The number of rules in \(\text{DD}(\text{KB})\) is at most exponential in \(|\text{KB}|\), the number of literals in each rule is at most polynomial in \(|\text{KB}|\), and \(\text{DD}(\text{KB})\) can be computed in time exponential in \(|\text{KB}|\).

The proof of this theorem can be found in [25]. It is very technical and if therefore out of scope of this deliverable, whose main focus is on hybrid reasoning.

**Adding DL-safe Rules.** The disjunctive program \(\text{DD}(\text{KB})\) can be combined with DL-safe rules by simply appending the rules to the program. The following theorem shows that \((\text{KB}, P)\) and \(\text{DD}(\text{KB}) \cup P\) entail the same ground facts about concepts and simple roles:

**Theorem 4.2.2 ([25]).** Let \(\text{KB}\) be a \(\text{SHIQ}(\text{D})\) knowledge base and \(P\) a DL-safe disjunctive datalog program. Then \((\text{KB}, P) \models \alpha\) if and only if \(\text{DD}(\text{KB}) \cup P \models_c \alpha\), where \(\alpha\) is a DL-atom \(A(a)\) or \(S(a, b)\) for a simple role \(S\), or \(\alpha\) is a ground non-DL-atom.

4.3 Reducing F-Logic to Disjunctive Datalog

In this section we show how to embed the other main formalism of our hybrid reasoning framework – F-Logic – into disjunctive datalog. The results presented here follow the general translation algorithm presented in [39]. For the full translation, we refer the reader to [39]; here we just overview the algorithm at a high level.
The major way for translating F-Logic into datalog is by flattening, which introduces so-called wrapper predicates representing certain relationships holding between object identifiers (OIDs). For example, the F-Logic statement

\[ \text{Person} : \text{John} \]

representing the fact that \textit{John} is an instance of \textit{Person} can be represented by the following statement:

\[ \text{isa(Person, John)} \]

In the above example the predicate \textit{isa} encodes the is-a relationship between two OIDs. Similarly, the subclass relationship can be represented using \textit{subclass} predicate, as in the following example:

\[ \text{Student :: Person} \leadsto \text{subclass(Student, Person)} \]

Single valued properties are encoded using \textit{fd} (functional data) predicate, and the multi-valued properties are encoded using \textit{mvd} (multi-valued data) predicate, as in the following examples:

\[
\begin{align*}
\text{mary}[\text{age} \rightarrow 30] & \leadsto \text{fd(}\text{age, mary, 30)} \\
\text{john}[\text{siblings} \rightarrow \{\text{bob, bill}\}] & \leadsto \text{mvd(}\text{siblings, john, bob}) \land \text{mvd(}\text{siblings, john, bill})
\end{align*}
\]

Finally, F-Logic rules can be encoded easily as datalog rules in the following way:

\[
\begin{align*}
x : \text{BadChild} & \leftarrow x : \text{Grandchild}[\text{hates} \rightarrow y, \text{sibling} \rightarrow y] \\
& \leadsto \text{isa(BadChild, x)} \leftarrow \text{isa(Grandchild, x)}, \text{mvd(hates, x, y)}, \text{mvd(sibling, x, y)}
\end{align*}
\]

It is important to understand that in the above translation we assume that function symbols are not used in the F-Logic knowledge base. This restriction is necessary to be able to handle F-Logic using disjunctive datalog. The original flattening transformation of F-Logic allows for functional terms; handling such terms will be possible once our framework is extended to handle rules with function symbols.

In [39] an important drawback of the above presented transformation scheme has been discussed: by using wrapper predicates, the indexing capabilities of the underlying engine become ineffective. Consider the \textit{isa} predicate, indexed by the first and the second position. By compiling a large number of is-a statements into the same predicate, one obtains a large index consisting of many entries. The search in such an index is slow and ineffective. Several strategies for remedying the situation have been presented in [39]. For the sake of brevity, we do not repeat these strategies here, but simply note that our framework is compatible with them.
4.4 Query Answering in Disjunctive Datalog

An efficient query answering algorithm is essential for practical applicability of our framework. Many techniques have been developed for disjunctive datalog without equality. These techniques can be used, provided that the usual congruence properties of equality are axiomatized correctly. This can be done by adding the following axioms to the program, where the last axiom is instantiated for each predicate occurring in the program [16]:

\[
\begin{align*}
x \approx x & \leftarrow HU(x). \\
x \approx y & \leftarrow y \approx x. \\
x \approx z & \leftarrow x \approx y, y \approx z. \\
P(\ldots, y, \ldots) & \leftarrow P(\ldots, x, \ldots), x \approx y.
\end{align*}
\]

Currently, the state-of-the-art technique for reasoning in disjunctive datalog is so-called intelligent grounding [15], and has been implemented successfully in the DLV disjunctive datalog engine. The algorithm is based on model building, which is performed by generating the ground instantiation of the program rules, generating candidate models, and using model checking algorithms to eliminate models which do not satisfy the ground rules. In order to avoid generating the entire grounding of the program, carefully designed heuristics is applied to generate the subset of the ground rules which have exactly the same set of the stable models as the original program. Query answering is reduced to model building, since \(A\) is not a certain answer if and only if there is a model not containing \(A\).

It is important to understand that in disjunctive datalog applications, computing the models is usually of more interest than query answering. For example, disjunctive datalog has been successfully applied to planning problems, where each plan is often “decoded” from a model.

On the contrary, the models of \(DD(KB)\) and other programs are of no interest. Furthermore, our programs do not contain non-monotonic negation. Hence, we propose query answering in disjunctive datalog by hyperresolution and basic superposition, which may be viewed as an extension of the fixpoint computation of plain datalog. A similar technique was presented in [9]. However, the algorithm presented there has two drawbacks: it does not take equality into account, and it does not specify whether application of redundancy elimination techniques is allowed.

4.4.1 Overview

We now informally present our query answering algorithm and later show formally its soundness and completeness. Let \(P\) be a positive datalog program and let \(Q\) be a query predicate not occurring in the body of a rule of \(P\). To compute all answers of \(P\) such that \(P \models_c Q(a)\), we saturate \(P\) by hyperresolution and perform paramodulation inferences between ground clauses to deal with equality. It is well-known that this calculus remains complete if ground literals are totally ordered under an arbitrary ordering \(\prec\), and inferences are performed on maximal literals only [6]. This ordering has the useful property that, in each ground disjunction, exactly one literal is maximal.
Hence, instead of performing an inference on each literal of a ground fact, it is sufficient to do so on the maximal literal only, which dramatically improves performance.

It is possible to show that, if all literals involving the predicate $Q$ are minimal w.r.t. $\prec$, then all ground consequences of $P$ will be derived as unit ground clauses in the saturation. In this way, to compute all answers to a query, it is sufficient to saturate $P$ only once using an appropriate ordering $\prec$. Thus our algorithm computes all consequences by a single saturation.

An example of a hyperresolution inference is presented in Figure 4.3. The maximal literals in the premises are underlined. Only these literals can participate in an inference. When performing hyperresolution, premises are matched to a disjunctive rule exactly as in the non-disjunctive case, the variables in the rule head are instantiated, and the remaining literals from the rule body are transferred to the rule head. Observe that, if premises and the rule are not disjunctive, then hyperresolution becomes exactly the least fixpoint operator used to evaluate non-disjunctive datalog programs. The consequences of the least fixpoint operator can be computed in polynomial time, so we get tractable behavior. In this way our algorithm supports the principle of “graceful degradation”: the user pays a performance penalty only for features actually used.

### 4.4.2 Formalization

We now formally present our query answering algorithm and show its correctness. In our algorithm, we use the well-known lexicographic path ordering (LPO) [12, 4], which is a term ordering induced over a precedence of function symbols $>_{P}$. Each LPO has the subterm property and, if $>_{P}$ is total, then LPO is total on ground terms. It is defined as follows: $s \succ_{lpo} t$ if

1. $t$ is a variable occurring as a proper subterm of $s$ or
2. $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$ and
   - (a) $f >_{P} g$ and, for all $i$ with $1 \leq i \leq n$, we have $s \succ_{lpo} t_i$ or
   - (b) $f = g$ and, for some $j$, we have $(s_1, \ldots, s_{j-1}) = (t_1, \ldots, t_{j-1})$, $s_j \succ_{lpo} t_j$, and $s \succ_{lpo} t_k$ for all $k$ with $j < k \leq n$ or
   - (c) $s_j \succ_{lpo} t$ for some $j$ with $1 \leq j \leq m$.

We are now ready to state our algorithm.

**Definition 4.4.1.** For a predicate symbol $Q$, let $\text{BS}^D_Q$ denote the $\text{BS}^D$ calculus parameterized in the following way:

- **Resolvent:** $R(a) \lor S(b) \lor V(c) \lor W(d)$
- **Rule:** $R(X) \lor S(Y) : \neg T(X,Y), U(Y)$
- **Facts:** $T(a,b) \lor V(c), U(b) \lor W(d)$

Figure 4.3: Hyperresolution Example
• All ground atoms of the form $Q(a)$ are smallest in the term ordering $\triangleright$ (for example, $\triangleright$ may be any LPO induced by a precedence in which $P \triangleright_P Q \triangleright_P T$ and $c \triangleright_P Q \triangleright_P T$, for any predicate symbol $P$ and a constant symbol $c$).

• All negative literals are selected.

Furthermore, for any two closures $s \simeq t \lor C$ where $s \simeq t$ is strictly maximal and $s \triangleright t$, and $Q(a) \lor D$ where $Q(a)$ is strictly maximal, $\mathcal{BS}_Q^D$ performs any possible superposition from $t$ into $Q(a)$, even if the corresponding position in $Q(a)$ is marked.

**Theorem 4.4.2.** Let $P$ be a positive satisfiable disjunctive datalog program and $Q$ a predicate not occurring in the body of any rule in $P$. Then $P \models_c Q(a)$ if and only if $Q(a) \in N$, where $N$ is the set of closures obtained by saturating $P$ under $\mathcal{BS}_Q^D$ up to redundancy.

**Proof.** $P \models_c Q(a)$ if and only if the set of closures $N'$, obtained as the result of saturating $P \cup \{ \neg Q(a) \}$ by $\mathcal{BS}_Q^D$ up to redundancy, contains the empty closure. Notice that, since all closures in $P$ are safe, all hyperresolvents are positive ground closures.

Consider first the case when no superposition inference is applied to the literal $\neg Q(a)$ in the saturation of $N'$. Since $P$ is satisfiable, $N'$ contains the empty closure if and only if a hyperresolution with $\neg Q(a)$ is performed in saturation. Since the literals containing $Q$ are smallest in the ordering, a positive literal $Q(a)$ can be maximal only in a closure $C = Q(a) \lor D$, where $D$ contains only literals with the $Q$ predicate. Since $\neg Q(a)$ is the only closure where $Q$ occurs negatively, if $D$ is not empty, no literal from $D$ can be eliminated by a subsequent hyperresolution inference. Hence, the empty closure can be derived from such $C$ if and only if $D$ is empty, which is the case if and only if $Q(a) \in N$.

Assume now that, in the saturation deriving $N'$, several negative superposition inferences from closures $a_i \simeq b_i \lor C_i$, $a_i \triangleright b_i$, are applied to $\neg Q(a)$, resulting in a closure $\neg Q(b) \lor C$, which then is resolved with a closure $Q(b) \lor D$, producing $C \lor D$. Such a derivation can be transformed into a derivation where superposition inferences are performed on $b_i$ into $Q(b) \lor D$, yielding $Q(a) \lor C \lor D$, which can then participate in a resolution with $\neg Q(a)$ to obtain $C \lor D$. Thus, we may successively eliminate each superposition into some $\neg Q(a)$ and obtain a derivation in which no superposition into $\neg Q(a)$ has been performed. Since in saturating $N$, all superposition inferences from the smaller side of the equality are performed into all literals containing $Q$, and all such inferences are sound, $Q(a) \in N$, so the claim of the lemma follows.

Assuming that $Q$ is a single predicate, or that it does not occur in the body of any rule in $P$, does not reduce the generality of the approach, as one can always add a new rule of the form $Q(x) \leftarrow A(x)$ to satisfy the conditions of Theorem 4.4.2.
5 SATISFACTION OF REQUIREMENTS

In this chapter we briefly summarize how our framework fulfills the requirements set forth in D1.1.

5.1 Supported Formalisms. Our framework supports all considered formalisms, namely description logic, F-Logic, disjunctive datalog and relational databases. However, some formalisms are not supported fully. In OWL-DL nominals are currently not supported; extending the translation is the main aspect of our future work. Currently, a well-known approximation can be used to provide sound, but incomplete reasoning [21]. In F-Logic the function symbols are currently not supported. However, including the support for function symbols is not difficult; the open problem is decidability of reasoning in such a case.

5.2 Level of Interoperability. Our framework does not inherently reduce any of the component formalisms to provide interoperability (apart from the points outlined above related to the future work). Instead of reducing target formalisms, in our framework the interface between the formalisms is reduced.

5.3 Decidability. Our framework supports decidable reasoning, so it fully satisfies the requirement.

5.4 Open- and Closed-World Assumption. Our framework does not yet support any non-monotonic features, so it currently supports only open world assumption. However, we anticipate that extending our algorithms to support non-monotonic reasoning will not be difficult, since disjunctive datalog has historically been mainly considered as a platform for such reasoning.

5.5 Support for Non-monotonic Negation. The same considerations as for the previous requirement considering non-monotonic reasoning apply to this requirement as well.

5.6 Support for Arbitrary Axioms. The usage of variables in DL-safe rules is not restricted. Hence, arbitrary axioms can be expressed using our framework, so the requirement is fulfilled completely.

5.7 Acceptable Performance Level. The evaluation of this requirement must be postponed until the framework will have been implemented and the performance comparison will have been conducted. However, our initial measurements from [31] are very promising. We strongly believe that optimization techniques for disjunctive datalog, such as the magic sets transformation [19] will provide an acceptable performance level.

5.8 Follow the Principle of “Graceful Degradation” The query answering algorithm presented in Section 4.4 has been designed precisely with this requirement in mind. Hence, this requirement has been satisfied from a theoretical point of view; it remains to be seen whether this requirement is satisfied in practice as well.
6 Conclusion and Future Work

In this deliverable we have presented a framework for hybrid reasoning in the Semantic Web. Our work is motivated by the need to integrate different formalisms currently considered for ontology modeling. OWL-DL and F-Logic are prominent examples of such languages, which are currently considered as the main formalisms for modeling semantically-enriched web services.

To achieve interoperability between different formalisms, we introduce the notion of DL-safe rules. Instead of reducing the component formalisms, we reduce the interface between them. As a consequence, rules apply only to individuals explicitly introduced in the ABox. We have discussed the effects of such a definition on a non-trivial example, which also shows that our approach increases the expressivity of its two components.

Our framework operationalizes hybrid reasoning by reducing target formalisms to disjunctive datalog with equality. A disjunctive datalog engine acts as the middleware, mediating between other target formalisms. To compensate for the increased computational complexity introduced by using disjunctive datalog, we developed a query answering algorithm, which, in case expressive features, such as disjunction, are not used, reduces to the least fixpoint operator. In this way our framework supports the principle of “graceful degradation”: the user does not pay a performance penalty for the features he does not use.

Description logics are supported in the framework by reducing DL knowledge bases to disjunctive datalog. Currently, $SHIQ(D)$ description logic is supported, which differs from OWL-DL mainly by not supporting nominals.

Furthermore, function-free F-Logic knowledge bases can also be encoded into disjunctive datalog, so our framework supports F-Logic as well. If F-Logic rules are DL-safe, they can be seamlessly integrated with a description logic knowledge base.

In our future work, we shall primarily focus on extending the framework to support all of OWL-DL. Furthermore, we shall attempt to extend the framework with some form of non-monotonic reasoning. We believe this is not going to be too difficult, since disjunctive datalog has been developed as a platform for non-monotonic reasoning, so a large body of research already exists.

Finally, in light of the discussion from D1.1, we shall consider approaches for integrating logic-based approaches to ontology management with database conceptual modeling, and thus provide a unifying framework for ontology representation, management and reasoning.
REFERENCES


