

DFL – A Dialog Based Integration of Concept and Rule Reasoners

Mira Balaban

Department of Information Systems Engineering

mira@cs.bgu.ac.il

(972)-7-6472222

Adi Eyal

Department of Computer Science

adie@cs.bgu.ac.il

Ben-Gurion University of the Negev

P.O.B. 653, Beer-Sheva 84105, Israel

Abstract

Description Logics (DLs) are subsets of First Order Logic, designed for reasoning about class based knowledge. Their expressive power is deliberately restricted, so to enable efficient inference. A DL reasoner or knowledge base is intended to be embedded as a special purpose component in a heterogeneous knowledge base. Therefore, the development of integration frameworks of DLs and other forms of reasoning is essentially important.

In this paper we introduce a formal scheme for the integration of information sources for which a combined declarative semantics is not available. The integration is defined by a syntactic compositional semantics, and implemented by a dialog process in which the independent reasoners make their failures public.

This scheme is used to formalize the integration of a DL reasoner with an expressive rule reasoner (for which a combined declarative semantics is not known). It is implemented in the *DFL* system, that integrates a DL reasoner and an F-Logic rule reasoner. The integrated system gives rise to a rich dialog between its components, since the DL inferences can trigger new rule inferences, and rule inferences can trigger new DL inferences. The *DFL* system is the first to support a true dialog between a DL and a rule reasoners, that operate under different semantical policies, e.g., the *Open World Assumption* for the DL reasoner, and the *Closed World Assumption* for the Rule reasoner. This architecture generalizes all existing hybrids of descriptions and rules.

Keywords: Integrated systems, description logics, rule base, Object-Oriented systems, F-Logic, compositional semantics, closed-world reasoning, open-world reasoning.

1 Introduction

Description Logic (DL) is a collective name for knowledge representation formalisms that concentrate on the management of essential descriptive vocabulary. It rests on the observation ([18]) that the natural ontology for providing information about a domain requires the ability to define, organize, and use intensional entities that stand for *concepts* and *roles* in a domain. Description logics were shown useful in Conceptual Modeling ([24, 3, 25]), Knowledge-Based applications ([37, 1, 26, 32]), and as view and query languages for data bases ([12, 21, 13, 22]).

The ontology of a description logic consists of a hierarchy of concepts and of roles. A concept is a set of objects and a role is a binary relation between objects. A description logic provides expressions called concept descriptions and role descriptions that characterize concepts and roles, respectively. These expressions are formed by description constructors that are built into the language. For example, the expression **and**(*plant*, **all**(*produces*, *mechanical_product*))¹ characterizes the concept “plants that *produce* only *mechanical_products*” and the expression **compose**(*produces*, *burried_at*) characterizes the role (binary relation) “the places in which the products produced by a plant are buried at”. A description logic knowledge base consists of a terminology, in which the concepts and roles are defined, and from factual information about concrete instances of concepts and roles. For example, the terminology can include a definition like *mechanical_plant* \doteq **and**(*plant*, **all**(*produces*, *mechanical_product*)) in which the necessary and sufficient conditions for membership in the concept *mechanical_plant* are defined as being a plant and producing only mechanical products. The structured nature of description expressions gives rise to questions such as subsumption between concepts, satisfiability of a terminology and membership in a concept.

Description logics are subsets of First Order Logic (FOL) (orthogonal to known subsets of FOL), designed for reasoning about class based knowledge ([15, 46, 61, 23]). Their expressive power is deliberately restricted, so to enable efficient inference. A DL reasoner or knowledge base is intended to be embedded as a special purpose component in a heterogeneous knowledge base. Therefore, the development of integration frameworks of DLs and other forms of reasoning is important.

The combination of a description logic reasoner with reasoners in different paradigms is studied and implemented within several research efforts. The KRYPTON system ([17]) is one of the early hybrids, combining a DL reasoner with a general purpose theorem prover ([56]). The CLASSIC ([15]) and BACK ([44, 45]) systems include an “operational” rule component. \mathcal{AL} -log ([27, 53]), and CARIN ([42, 43]) extend a Horn clause reasoner with a DL reasoner. Alternatively, there are numerous efforts to extend description logics to account for common sense features such as default and epistemic reasoning ([7, 29, 51]), to account for domain extensions (e.g., time [2], space [33]), to add features such as *aggregates* ([4]), and to combine a DL reasoner with concrete domains ([5]). Nevertheless, there is no general framework for embedding a description logic reasoner as a component within a heterogeneous knowledge base.

¹Written also *plant* $\sqcap \forall$ *produces.mechanical_product*.

1.1 Integration of Description Logics with Rules

The integration of description logics with rules is particularly interesting since they provide for efficient reasoning with restricted subsets of first order logic that complement each other. Description logics are designed to model, in a declarative manner, restricted vocabularies of concepts that demonstrate complex hierarchies, while rules model contingent directional dependencies. In the database jargon rules are called *triggers* or *Event-Condition-Action (ECA)* rules ([60]). When the event of the trigger occurs, and if the condition holds, then the action part is executed (the rule is fired).

Providing a uniform semantics to a Description Logic – Rule hybrid is hard since the first has a classical monotonic semantics while the latter has non-monotonic semantics². In particular, rules dictate logical implication only in the positive direction (from conditions to consequences), and not in the contra-positive direction (the negation of the consequence entails the negation of the condition). For example, the rule

If it is known that all products of a plant are consumed by rich companies then the plant is considered to be a good investment, which is written
 $good_investment(P) \leftarrow P \in plant, P \in \mathbf{all}(produces, \mathbf{some}(consumer, rich_company))$.

can be used only to conclude that an object is a good investment, provided that it is known to be a plant that all of its products are consumed by rich companies (*plant* and *rich_company* denote concepts, *produces* and *consumer* denote roles, and *good_investment* denotes a regular database relation). If the negation of the consequence is given, it cannot lead to the conclusion that one of the rule conditions does not hold. For that reason, rule mechanisms usually have a special kind of negation (negation as failure), used only in the condition component of a rule.

The three major efforts to combine rule and description reasoning include the CLASSIC and BACK systems, the epistemic logic \mathcal{ALCK} ([29]), and the \mathcal{AL} -log and CARIN frameworks. CLASSIC and BACK include a rule component that acts like a database triggering system. The rules describe membership implications that can trigger further Abox inference, and vice-versa. Both systems compute an Abox which is the transitive closure of their combined entailment relation in a *dialog* like manner (up to contradictions that terminate the propagation). The epistemic description logic \mathcal{ALCK} provides declarative semantics to the CLASSIC and BACK integration of rules and descriptions. The \mathcal{AL} -log and CARIN frameworks strengthen a negation-free DATALOG rule component with description logic reasoning, using a constraint logic programming approach. They assume an ontology of facts, concepts and roles, and support conditional factual knowledge that depends on Abox formulae. In particular, they support only “one-way” integration (to distinguish from a *dialog* between components) since a description logic reasoner is used as a constraint system on top of a DATALOG reasoner.

Observing the DL-Rule integration frameworks we see that either they support smooth integration with a restricted rule component, or they extend a rule component with DL capabilities. In particular, there is no integration with a powerful rule language, and there is no consideration

²In the artificial intelligence jargon, the classical monotonic semantics is termed *Open World Assumption (OWA)* and the rule style non-monotonic semantics is termed *Closed World Assumption (CWA)*.

of the integrated system properties from a software engineering viewpoint. \mathcal{DFL} tries to cope with these aspects. The desired DL-Rule integration is enabled by a rule language that includes DLs as an integral part, and can reason with and about descriptions and description formulae. The flexible structure is achieved by a *compositional architecture*.

1.2 The \mathcal{DFL} System

\mathcal{DFL} is an implemented system that enables smooth integration of any description logic reasoner with a powerful rule reasoner. The rule component supports terminological reasoning about concepts, membership reasoning about individuals, and other factual knowledge. In particular, the rules enable meta-level reasoning about concepts, such as various classifications of the concept and role sets (e.g., reliable concepts, singleton concepts, etc). The \mathcal{DFL} integration is compositional in the sense that it provides a *plug-in architecture* for knowledge components that reason about a terminology of concepts, roles and their instances. Following the standard of multi-tier systems, \mathcal{DFL} separates the knowledge base of concept, role and individuals from the application logic reasoners. That is, the reasoners access a single knowledge base (through an administrator that is responsible for management and retrieval alone). The semantics of \mathcal{DFL} is defined in a compositional manner from the semantics of its components.

The rule language in \mathcal{DFL} is *F(roles)-Logic* ([38, 39]), an expressive object-oriented logic. F-Logic can reason freely about concepts and instances since the description logic semantics can be identified with the object-oriented one ([9]). Under this view, concepts and their instances are entities, to which constructors can be applied, and on which description formulae can be evaluated³. Indeed, the integrated system gives rise to a rich dialog between its components, since the DL inferences can trigger new rule inferences, and rule inferences can trigger new DL inferences.

Compositionality is a desirable property of multi-component systems. It means that the semantics of the system can be derived from the independent semantics of its components. That is, for a system with n components C_1, \dots, C_n , and an integration operator \circ , its semantics $S(\circ(C_1, \dots, C_n))$ is compositional with respect to \circ , if it can be derived from $S(C_1), \dots, S(C_n)$ ($S(C_i)$ stands for the semantics of C_i). Clearly, it is implied that for components C, C' with the same semantics (i.e., $S(C) = S(C')$), $S(\circ(C, C_1, \dots, C_n)) = S(\circ(C', C_1, \dots, C_n))$ for any C_1, \dots, C_n . Therefore, the least Herbrand model semantics is not compositional with respect to union of sets of formulae. Compositionality is important both for theoretical and for practical reasons since it allows for inductive semantic analysis and enables the construction of modular systems, with plug-in components. Compositionality is a standard approach in linguistics and in the theory of programming languages.

The semantics of \mathcal{DFL} is defined in a compositional manner from the semantics of its components. It is given by a set of syntactic objects, that stands for the intended meaning of the integrated system. Each component has its own syntactic semantics (syntactic transformation), and the compositional semantics is defined as a fixed point of the union of the component syntactic transformations. The \mathcal{DFL} system implements a compositional architecture that leads to

³Interestingly, the uniform view of object-oriented and description logics leads to some free extensions of the rules of \mathcal{AL} -log and CARIN, i.e., extensions that do not complicate their functionality.

complete focussed behavior with respect to the compositional semantics. In particular, each component preserves its autonomous status and operates under its own semantical policy, i.e., the *Open World Assumption (OWA)* for the description logic reasoner, and the *Closed World Assumption (CWA)* for the rule reasoner. Moreover, each component can be pre-compiled and optimized using techniques that are appropriate for its paradigm. Additional components that implement other reasoning techniques, or provide for other knowledge paradigms can be easily added. In that sense the system is truly an open, plug-in system. The \mathcal{DFL} integration framework can be generalized into a general compositional architecture for heterogeneous knowledge bases.

The \mathcal{DFL} compositional semantics captures the transitive closure style operation of the CLASSIC and BACK systems. Its expressive power is larger since its rules are not limited to membership implications, and it can support any description logic reasoner. It is not comparable with \mathcal{AL} -log, CARIN and \mathcal{ALCK} since although its rule component can process any kind of rules, it suffers from the inherent limitations of rule systems, i.e., the inability to handle case analysis since they process each rule in isolation. Indeed, this is the “price” payed for the open compositional architecture. Yet, the novelty of the \mathcal{DFL} system is its being the first open integration of description logics and rules, that is backed by a formal compositional semantics.

Description logics are introduced in Section 2. In Section 3 F-logic is used to provide an object-oriented view of concepts and roles. In Section 4 we describe the \mathcal{DFL} integration framework, including its architecture, compositional semantics, and reasoning procedure. Section 5 describes the generalization of \mathcal{DFL} into a general compositional integration scheme. Section 6 is the conclusion and discussion of future research. The implemented \mathcal{DFL} system is described in Section A, and a \mathcal{DFL} knowledge base and query answering is presented in Appendix B.

2 Description Logics – a Terminology Oriented View of Concepts and Roles

Description logics form a spin-off of the KL-ONE school ([19, 62]), that concentrate on the management of class-based domain terminology. DLs emphasize the importance of direct, well defined semantics, and of a limited and efficient inferential service. A DL terminology consists of *concepts* and *roles*, which stand for subsets and binary relations over a domain of individuals, respectively. The concepts form a *taxonomy*, i.e., a partially ordered structure that derives from the lattice of subsets.

A DL language consists of symbols for concepts, roles and individuals, and a small set of description constructors. *Descriptive terms* are formed by applying the constructors to concept/role symbols. For example, the concept of a *mechanical_plant*, i.e., a plant that produces only mechanical products, can be described by the concept term:

$$\mathbf{and}(\textit{plant}, \\ \mathbf{all}(\textit{produces}, \textit{mechanical_product}))$$

where, **and** and **all** are concept description constructors, *plant* and *mechanical_product* are *concept symbols*, and *produces* is a *role symbol*. The concept of a *uniform_plant*, i.e., a plant that produces

a single kind of products, can be described by the concept term:

$$\mathbf{and}(\mathit{plant}, \\ \mathbf{atleast}(\mathit{produces}, 1), \\ \mathbf{atmost}(\mathit{produces}, 1))$$

In this term **atleast** and **atmost** are concept description constructors. The binary relation **produced-by** can be described by the role term **inv**(*produces*), i.e., the inverse relation of *produces*. Here **inv** is a role description constructor.

Description formulae are either terminological formulae or membership formulae. Terminological formulae define or describe concepts and roles. For example,

$$\mathit{mechanical_plant} \doteq \mathbf{and}(\mathit{plant}, \\ \mathbf{all}(\mathit{produces}, \mathit{mechanical_product}))$$

is a *definition formula* for the concept *mechanical_plant*, and

$$\mathit{uniform_plant} \leq \mathbf{and}(\mathit{plant}, \\ \mathbf{atleast}(\mathit{produces}, 1), \\ \mathbf{atmost}(\mathit{produces}, 1))$$

is an *inclusion formula* for the concept *uniform_plant*. The latter states that every uniform plant produces exactly one product but not necessarily vice-versa, i.e., not every plant that produces a single product is necessarily a uniform plant. Membership formulae describe memberships of individuals in concepts or in roles. For example,

$$w \in \mathbf{and}(\mathit{waste}, \mathit{radioactive_material}) \\ (\mathit{pl}, \mathit{pr}) \in \mathbf{androle}(\mathit{produces}, \mathit{consumes})$$

state that *w* is a radioactive waste, and that *pr* is both produced and consumed by *pl*. In general, every set of constructors defines a DL language.

A DL knowledge base consists of two distinguished components termed *Tbox* and *Abox*. The *Tbox* is a schema level intensional component, that defines a *terminology* of concepts and roles. The *Abox* is a data level extensional component, that introduces individual members of concepts, and role relationships that hold between individuals. The *Tbox* can include *equations* (\doteq) or *inclusions* (\leq) between concept/role terms. Equation formulae provide necessary and sufficient conditions for membership in a concept or a role, and inclusion formulae provide only necessary conditions. Concept/role symbols that appear on the left side of an equation formula are called *defined concepts/roles*, and concept/role symbols that appear on the left side of an inclusion formula are called *primitive concepts/roles*. The overall structure of a typical DL knowledge base is given in Figure 1. Table 1 describes a DL knowledge base that deals with the ecological aspect of industrial plants. The *Tbox* includes 8 concept inclusion formula, 3 role inclusion formulae, and 2 concept definition formulae. The *plant* concept is introduced as a primitive concept, whose members can be *located_at* only in *places*. The *produces* role is introduced as a primitive role, that can hold only between *plants* to *products*. The *mechanical_plant* concept is defined as a plant that *produces* only *mechanical_products*, and the *risky_place* concept is defined as a place where some *toxic_waste* is *buried_at*. The description language used to build these formulae is {**and**, **all**, **some**, **domain**, **range**, **inv**}. The meaning of these constructors is defined formally below.

D

<i>Tbox</i>		<i>Abox</i>	
<i>EQUATIONS</i>	<i>INCLUSIONS</i>	<i>C members</i>	<i>R members</i>
$c_1 \doteq c_2$	$c_1 \leq c_2$	$o \in c$	$(o_1, o_2) \in r$
$r_1 \doteq r_2$	$r_1 \leq r_2$		

Figure 1: The Structure of a Typical DL KB.

Kind	No.	Description in words	description formula
Tbox			
Primitive-concepts	t1)	<i>product is_a top</i>	$product \leq top$
	t2)	<i>place is_a top</i>	$place \leq top$
	t3)	<i>waste is_a product</i>	$waste \leq product$
	t4)	<i>radioactive_material is_a product</i>	$radioactive_material \leq product$
	t5)	<i>plant is_a top, located_at somewhere</i>	$plant \leq \mathbf{and}(top, \mathbf{some}(located_at, top))$
	t6)	<i>dangerous_plant is_a plant</i>	$dangerous_plant \leq plant$
	t7)	A <i>radioactive_material</i> that is also a <i>waste</i> , is a <i>toxic_waste</i>	$\mathbf{and}(radioactive_material, waste), \leq toxic_waste$
Primitive-roles	t8)	<i>produces</i> is a relation between objects and <i>products</i>	$produces \leq \mathbf{range}(product)$
	t9)	A <i>product</i> may be <i>buried_at</i> a <i>place</i>	$buried_at \leq \mathbf{and}(\mathbf{domain}(product), \mathbf{range}(place))$
	t10)	<i>located_at</i> is a relation between objects and <i>places</i>	$located_at \leq \mathbf{range}(place)$
Defined-concepts	t11)	A <i>mechanical_plant</i> is a <i>plant</i> that <i>produces</i> only <i>mechanical_products</i>	$mechanical_plant \doteq \mathbf{and}(plant, \mathbf{all}(produces, mechanical_product))$
	t12)	A <i>risky_place</i> has some <i>toxic_waste</i> buried in it	$risky_place \doteq \mathbf{some}(\mathbf{inv}(buried_at), toxic_waste)$
Abox			
Concept-members	a1)	<i>pl</i> is a mechanical plant	$pl \in mechanical_plant$
	a2)	<i>w</i> is a radioactive waste	$w \in \mathbf{and}(waste, radioactive_material)$
Role-members	a3)	<i>pl</i> produces a product <i>pr</i>	$(pl, pr) \in produces$
	a4)	<i>w</i> is buried at <i>dp</i> (dump)	$(w, dp) \in buried_at$
	a5)	<i>pl</i> is located at <i>dp</i>	$(pl, dp) \in located_at$

Table 1: A Description Logic Knowledge Base: Tbox, Abox.

2.1 Formal Definition of Description Logics

2.1.1 Syntax

Symbols: A description logic includes concept name symbols (c_n), role name symbols (r_n), object symbols (o), two special concept symbols *top* and *bottom*, and a finite set of concept and role constructors.

Terms: The terms of a description logic are either *concept terms*, or *role terms*. Concept/role terms are all concept/role symbols, and all syntactically legal applications of a concept/role constructors to terms, respectively. A concept term of any kind is denoted c , and a role term of any kind is denoted r .

Formulae: The Tbox formulae are either *equations* $c_1 \doteq c_2$, $r_1 \doteq r_2$, or *inclusions* $c_1 \leq c_2$, $r_1 \leq r_2$. The left side is usually restricted to be a symbol. In that case, equations are termed *definitions*, and inclusions are termed *primitive specifications*. Most DLs do not allow role definitions at all. The Abox formulae are *memberships* $o \in c$, $(o_1, o_2) \in r$.

2.1.2 Semantics

Meaning of formulae is given by a set theoretic semantics. An *interpretation* \mathbf{I} is a pair (D, Υ) , of a domain D and an interpretation function Υ , such that concept symbols are assigned subsets of D , role symbols are assigned binary relations over D , object symbols are assigned elements of D , $\Upsilon(\text{top})$ is D , and $\Upsilon(\text{bottom})$ is \emptyset . The interpretation function Υ is augmented to all concept/role terms by building into it a fixed meaning for each description constructor. For example, for the DL $\{\mathbf{and}, \mathbf{all}, \mathbf{some}, \mathbf{at-most}, \mathbf{at-least}, \mathbf{domain}, \mathbf{range}, \mathbf{inv}, \mathbf{and-role}\}$, the meaning of concept terms is defined as follows:⁴

$$\begin{aligned}
 \Upsilon(\mathbf{and}(c_1, c_2)) &= \Upsilon(c_1) \cap \Upsilon(c_2) \\
 \Upsilon(\mathbf{all}(r, c)) &= \{d \in D \mid \Upsilon(r)(d) \subseteq \Upsilon(c)\} \\
 \Upsilon(\mathbf{some}(r, c)) &= \{d \in D \mid \Upsilon(r)(d) \cap \Upsilon(c) \neq \emptyset\} \\
 \Upsilon(\mathbf{at-most}(r, n)) &= \{d \in D \mid \#\Upsilon(r)(d) \leq n\} \\
 \Upsilon(\mathbf{at-least}(r, n)) &= \{d \in D \mid \#\Upsilon(r)(d) \geq n\} \\
 \Upsilon(\mathbf{and-role}(r_1, r_2)) &= \Upsilon(r_1) \cap \Upsilon(r_2) \\
 \Upsilon(\mathbf{domain}(c)) &= \{(d, e) \in D \times D \mid d \in \Upsilon(c)\} \\
 \Upsilon(\mathbf{range}(c)) &= \{(d, e) \in D \times D \mid e \in \Upsilon(c)\} \\
 \Upsilon(\mathbf{inv}(r)) &= \{(d, e) \in D \times D \mid (e, d) \in \Upsilon(r)\}
 \end{aligned}$$

Satisfaction of formulae in an interpretation is defined by interpreting \doteq as set equality, \leq as set inclusion, and \in as membership over D . An interpretation is a *model* for a set of formulae if it satisfies all formulae. A formula γ is logically implied from a terminology T , i.e., $T \models \gamma$, if it is satisfied in every model of the terminology. A concept term t is *coherent* with respect to a terminology T , if there exists a model (D, Υ) of T , such that $\Upsilon(t) \neq \Phi$. Term t_1 is subsumed by

⁴We view relations as set-valued functions, whenever it simplifies the presentation. For a role symbol r and a domain element d , $\Upsilon(r)(d)$ denotes the set of all domain elements related to d via $\Upsilon(r)$, i.e., the set $\{e \mid (d, e) \in \Upsilon(r)\}$.

term t_2 ($t_1 \sqsubseteq t_2$) in a terminology T , if in every model (D, Υ) of T , $\Upsilon(t_1) \subseteq \Upsilon(t_2)$. *Equivalence* of terms is defined as two way subsumption.

2.1.3 Reasoning in Description Logics

Description logics can reason about concepts/roles and about individuals. The first kind, termed *Tbox reasoning*, deals with queries about inter-relationships between concepts, and about properties of concepts. The second kind, termed *Abox reasoning*, answer queries about membership of individuals in concepts and roles. The subsumption relationship plays a major role in Tbox reasoning, as most queries can be phrased as subsumption queries. A query whether a concept term c is coherent in a terminology can be rephrased as the subsumption query $? c \sqsubseteq \text{bottom}$. The query:

Is a *plant* necessarily *located_at* a place?

is rephrased as the subsumption query:

$? \text{plant} \sqsubseteq \mathbf{all}(\text{located_at}, \text{place})$

The answer, when applied to Table 1 is *true*, since the range of the role *located_at* is *place*, and a *plant* is *located_at* somewhere. Similarly, the Tbox of Table 1 implies the subsumption $\text{risky_place} \sqsubseteq \text{place}$, since the domain of the role $\mathbf{inv}(\text{buried_at})$ is *place*. Abox reasoning with Table 1 implies $pr \in \text{mechanical_product}$, and $dp \in \text{risky_place}$. The first results from pl being an instance of *mechanical_plant*, and producing pr . The latter holds since w is a *toxic_waste* ((a2), (t7)) that is *buried_at* dp (a4).

Description logics support powerful reasoning due to their expressive schema languages. The description constructors enable schema level and data level reasoning which is beyond the capabilities of object-oriented or deductive databases. In particular, they can reason with *incomplete knowledge*. This is due to constructors such as **not**, **some**, **or**, **at-least**. For example, if the Abox includes a single assertion $p \in \text{risky_place}$ then together with the Tbox of Table 1 it implies that the concept *toxic_waste* is not empty even though no concrete individual is known for that concept. Likewise, if in Table 1 we have also the definition of a *safe_place* as a place where no toxic_waste is buried at, i.e., $\text{safe_place} \doteq \mathbf{all}(\mathbf{inv}(\text{buried_at}), \mathbf{not}(\text{toxic_waste}))$, then together with the single Abox assertion $p \in \text{place}$ it implies $p \in \mathbf{or}(\text{risky_place}, \text{safe_place})$.

As demonstrated in the above examples, DL reasoning is not dictated by temporary population in a knowledge base, but follows from the meaning of the description constructors. In that sense, DLs do not operate under the *Closed World Assumption (CWA)*, which dominates most reasoning approaches in Artificial Intelligence and in deductive databases.. For example, the above query “Is a *plant* necessarily *located_at* a place?”, in a typical AI or database context, would have been formulated as a query about the individual members of *plant*:

“Are all objects to which a *plant* is related by the role *located_at*, members of the *place* concept?”

DLs can be understood as operating under the *Open World Assumption (OWA)*, where a momentary population in a knowledge base does not provide sufficient evidence on essential definitions in the terminology.

There are two main approaches for reasoning in description logics. The earlier approach, termed *structural subsumption*, concentrates on subsumption computation. The computation is based on rules that perform structural comparisons of concept terms. For example, for the **and** concept constructor, there would be a rule like:

$$\mathbf{and}(c_1, \dots, c_m) \sqsubseteq \mathbf{and}(d_1, \dots, d_n) \longleftarrow \begin{array}{l} \exists \{c_{i_1}, \dots, c_{i_n}\} \subseteq \{c_1, \dots, c_m\} \\ \text{such that } c_{i_j} \sqsubseteq d_j, 1 \leq j \leq n. \end{array}$$

For the **all** concept constructor there might be a rule like:

$$\mathbf{all}(r_1, c_1) \sqsubseteq \mathbf{all}(r_2, c_2) \longleftarrow c_1 \sqsubseteq c_2 \text{ and } r_2 \sqsubseteq r_1.$$

In this method, the description terms are first expanded by substituting all defined concepts or roles by their defining terms. Therefore, structural subsumption cannot account for cyclic definitions. The performance is also strengthened by applying some normalization transformations to the terms, so to remove syntactical differences that are semantically meaningless. Most structural subsumption algorithms are incomplete. The CLASSIC ([15]) and BACK ([44, 45]) systems are based on structural subsumption.

A different, Tableaux calculus based approach ([55], KRIS [6], CRACK [20], FaCT [36]), reduces subsumption to the problem of coherency. Constraint propagation rules, that result from the semantics of the constructors, are used to construct a small number of generic finite models that are sufficient for determining contradiction. The presence of contradiction implies that the problem is incoherent, and the lack of contradiction implies that the problem is satisfiable. Constraint-based algorithms can handle cyclic terminological definitions and are complete.

Major efforts were devoted to the study of the subsumption problem in different description logics ([41, 50, 54, 49, 28]), to the study of subsumption/classification algorithms in existing systems ([8, 47]), and to the development of subsumption algorithms ([52, 35, 55]). These research efforts show that in expressive description logics subsumption is not tractable. These results emphasize the relevance of frameworks for combination of DLs with other forms of knowledge.

2.2 Integration of Description Logics with Rules

A full integration of description and rule reasoning is realized in the CLASSIC and BACK systems ([15, 44, 45]), and formalized in the epistemic logic \mathcal{ALCK} ([29]). The rules are restricted to describe membership implications like

If a product is known to be a toxic waste then all of its burial places are risky, which is written

$$P \in \mathbf{all}(\mathit{buried_at}, \mathit{risky_place}) \longleftarrow P \in \mathit{toxic_waste}$$

. On the events of object insertion or deletion the appropriate rules are fired and propagate the new membership information that is implied from the rules. For example, if o_1 is inserted to *toxic_waste*, then by the above rule o_1 is also a member of $\mathbf{all}(\mathit{buried_at}, \mathit{risky_place})$. Now, if it is also known that o_1 stands in the relation *buried_at* with o_2 , then the description logic component

concludes that o_2 is a member of *risky_place*. If there is a similar trigger on this latter concept it is fired as well.

The rules are not applied in the contra-positive direction. That is, if the Abox includes also the assertion $o_3 \in \mathbf{not}(\mathbf{all}(\mathit{buried_at}, \mathit{risky_place}))$, then the rule does not lead to the assertion $o_3 \in \mathbf{not}(\mathit{toxic_waste})$. Likewise, the rules are not made part of the terminology, and do not affect Tbox reasoning. For example, a query $\mathit{toxic_waste} \sqsubseteq \mathbf{all}(\mathit{buried_at}, \mathit{risky_place})$ to the above database succeeds only if the concept description *toxic_waste* can be classified as subsumed by the concept description $\mathbf{all}(\mathit{buried_at}, \mathit{risky_place})$, (although the above rule classifies every member of *toxic_waste* also as a member of $\mathbf{all}(\mathit{buried_at}, \mathit{risky_place})$).

The CLASSIC and BACK systems compute the transitive closure of their combined entailment relation in a *dialog* like manner (up to contradictions that terminate the propagation). In both systems the rule component is added operationally. They lack a formal – declarative or procedural – combined semantics. A declarative semantics that fully captures their operation is given in *ALCK*.

The epistemic logic *ALCK* extends a description logic with an epistemic operator that accounts for all “known” members of a concept. The logic can describe a “weak inclusion” relation, written $KC \sqsubseteq D$ (K is the epistemic operator, and C and D are concept descriptions), which states that all known members of C are members of D . That is, those individuals that are necessarily in the extensions of C in all models are also in any extension of D . Weak inclusion does not imply the contra positive implication since a member of $\mathbf{not}D$ that is a member of C does not falsify the weak inclusion. Weak inclusion captures the intuition and mode of operation of procedural rules that describe membership implications as in CLASSIC and BACK. A membership implication rule as above is written

$$K\mathit{toxic_waste} \sqsubseteq \mathbf{all}(\mathit{buried_at}, \mathit{risky_place})$$

and interpreted as “all *known* members of *toxic_waste* are members of $\mathbf{all}(\mathit{buried_at}, \mathit{risky_place})$ ”. Therefore, the contra-positive direction does not apply, since an object that is not in $\mathbf{all}(\mathit{buried_at}, \mathit{risky_place})$ can still be a member of *toxic_waste*. Indeed, in [29] it is shown that *ALCK* accounts for the combined DL-Rule operation of CLASSIC and BACK.

The *AL*-log and CARIN frameworks ([27, 53, 42, 43]) follow the constraint logic programming approach in strengthening a negation-free DATALOG rule component with description logic reasoning. Both systems provide declarative semantics to the integrations. The DATALOG atoms are distinguished from the description logic atoms, and rule consequences are restricted to logic atoms alone. Therefore, the description logic negation cannot apply to rule consequences and reasoning in the contra-positive direction is blocked. In these frameworks there is no room for a *dialog* between the components since the description logic reasoner is used as a constraint system on top of a DATALOG reasoner. A special strength of *AL*-log and CARIN is their capability to reason with cases about description assertions that are specified in rule bodies.

Careful observation of the declarative semantics of *AL*-log, CARIN, and *ALCK* shows that they are not compositional. For example, consider two CARIN rule components:

1. $R_1 = \{p(X) \leftarrow X \in c\}$,
2. $R_2 = \emptyset$,

and the assertions component $A = \{o \in d\}$. Clearly, the least Herbrand model of both components is $\{o \in d\}$. Yet, when combined with the description logic component $DL = \{c \doteq d\}$ that defines the concept c to be equal to concept d , they yield different semantics. The semantics of (R_1, DL, A) is $\{o \in d, o \in c, p(o)\}$ while the semantics of (R_2, DL, A) is $\{o \in d, o \in c\}$.

The following observation is influenced from the smooth integration of DLs and rules in \mathcal{DFL} , where rules can include terminological formulae as regular predications.

Claim 2.1 *The integration frameworks of \mathcal{AL} -log and CARIN can be extended to include concept inclusion atoms in rule bodies, without affecting the results of these frameworks.*

Proof: Sketched. The inference algorithms in both frameworks operate by developing tableaux branches for various combinations of description constraints in the rules, and testing them for satisfiability or logical implication. The tableaux branches can include inclusion constraints. Hence, the occurrence of inclusion description constraints in the rules does not add any complication. \square

3 An Object-Oriented View of Concepts and Roles

In this section we show that a description logic data model of concepts and roles formed by given description constructors, can be viewed as a typical object-oriented data model, in which both individuals and concepts are entities, and description constructors play the role of *class constructors* (to distinguish from the more common *instance constructors*). This view is markedly different from the standard first order view of concepts and roles as unary and binary predicates. It enables reasoning about and with descriptions since they are the terms of the logic, not the predicates. Following this view, we use the object-oriented logic F-Logic to write rules about description formulae of inclusion and membership.

3.1 DLs – as sorted F-Logic Languages

Let P be a (finite) set of description constructors. We define L_P , a description language with description constructors in P , as a sorted subset of F-Logic ([38, 39]). The definition is independent of P .

3.1.1 Syntax

1. **Symbols:**

P – a finite set of *description constructors*.

C – a set of *concept symbols*. $\{top, bottom\} \subset C$.

R – a set of *role symbols*.

O – a set of *individual symbols* (also called *object symbols*).

S – a set of three *sort symbols*: $\{concept, role, individual\}$.

Ψ – a *sort assignment*. $\Psi : C \rightarrow \{concept\}$, $\Psi : R \rightarrow \{role\}$, $\Psi : O \rightarrow \{individual\}$. An n -ary constructor in P is assigned a sort in the form of an $n + 1$ tuple over S . For example,

$\Psi(\mathbf{all}) = (\text{role}, \text{concept}, \text{concept})$, since the **all** constructor takes a role and concept descriptions as arguments and yields a new concept description, as in $\mathbf{all}(r, c)$.

2. **Well-Sorted Terms:** All concept, role, and object symbols are terms; their sorts are given by Ψ . Complex terms are formed by well sorted applications of description constructors to terms. A complex term $\mathbf{op}(t_1, \dots, t_n)$, where $\Psi(\mathbf{op}) = (s_1, \dots, s_{n+1})$, is well sorted if the sort of t_i ($1 \leq i \leq n$) is s_i . The sort of the term is s_{n+1} . Below we use c, r, o to denote concept, role, and individual (object) terms, respectively.

3. **Formulae:**

- $c_1 \doteq c_2; \quad c_1 \leq c_2; \quad o \in c;$
- $\forall_{ind} X, Y, (X[r_1 \rightarrow \{Y\}] \equiv X[r_2 \rightarrow \{Y\}]);$ ⁵ Syntactic shortcut: $r_1 \doteq r_2$.
The intuitive reading of this formula is: “For all individuals X and Y , Y is related to X by r_1 (or Y is in the values of r_1 on X) if and only if Y is related to X by r_2 ”.
- $\forall_{ind} X, Y, (X[r_1 \rightarrow \{Y\}] \rightarrow X[r_2 \rightarrow \{Y\}]);$ Syntactic shortcut: $r_1 \leq r_2$.
The intuitive reading of this formula is: “For all individuals X and Y , if Y is related to X by r_1 then Y is related to X by r_2 ”.
- $o_1[r \rightarrow \{o_2\}];$ Syntactic shortcut: $(o_1, o_2) \in r$.
The intuitive reading of this formula is: “The individual o_2 is related to o_1 by r ”.

Note that membership formulae, and concept inclusion and equality formulae, are understood as atoms of F-Logic, while role inclusion and equality are understood as syntactically distinguished rules and equivalence formulae of F-Logic. Clearly, a variant of F-Logic, that has built-in inclusion and membership relations for roles, might be more appropriate. In such a variant, all description formulae will be atoms of the logic.

3.1.2 Semantics

The semantics of L_P , as an F-Logic language, is defined over a partially ordered domain U , with a greatest and a least elements, where terms are mapped to elements of U , and the symbols *top* and *bottom* are assigned the greatest and least elements, respectively. Formulae are interpreted by interpreting \doteq and \leq between concept terms as equality and the partial ordering, respectively; \in between an object symbol and a concept term is interpreted as the membership binary relation over U ; \doteq and \leq between role terms are interpreted as assertions about equality and implication of methods, respectively; \in between a pair of object symbols and a role term is interpreted as an assertion about a value of a method.

The intended meaning of the description operators is given by an F-Logic theory FL_P , called *the corresponding theory* for L_P . FL_P is not part of L_P . Its main property is that it provides

⁵The subscribed quantifier “ \forall_{ind} ” quantifies over the sort of individuals.

equivalence with the standard set-theoretic semantics of description languages. That is, for every set of formulae Γ , and a formula γ in L_P :

$$\Gamma \stackrel{DL}{\models} \gamma \quad \text{iff} \quad FL_P, \Gamma \stackrel{FL}{\models} \gamma \quad ,$$

where $\stackrel{DL}{\models}$ denotes logical implication under the set-theoretic semantics of description languages, and $\stackrel{FL}{\models}$ denotes logical implication in F-Logic. For further details consult [9], where a corresponding theory for $P = \{\mathbf{and}, \mathbf{all}, \mathbf{at-least1}, \mathbf{and-role}\}$ is given. A major result of [9] is that given a corresponding theory FL_P to L_P , the semantics of F-Logic provides a full account to L_P , i.e., it correctly simulates logical implication and subsumption relations, while preserving the direct semantics.

Corollary 3.1 *Let t_1, t_2 be terms of L_P , and FL_P an F-Logic theory that corresponds to L_P . Then, $\Gamma \stackrel{DL}{\models} t_1 \sqsubseteq t_2$ iff $FL_P, \Gamma \stackrel{FL}{\models} t_1 \leq t_2$.*

3.2 F-logic Rules About Concepts and Roles

The semantics that F-Logic assigns to descriptions is different from the standard first order logic view. The common approach is to view concept and role symbols as unary and binary predicates. That is, the membership formulae $o \in c$ and $(o_1, o_2) \in r$ are interpreted as unary and binary predications $c(o)$ and $r(o_1, o_2)$. Therefore, DL-Rule systems allow membership formulae, ground or not, in the rule. However, no DL-Rule framework allows for inclusion or equality formulae in the rules. F-Logic views descriptions as denoting concept or role entities. Therefore, the membership and inclusion predicates of DLs are also predicates of F-Logic. As a result, rules can reason about DL formulae, ground or not. Moreover, descriptions can be given as arguments to predicates, thereby enabling meta-level reasoning about concepts and roles.

A rule is a formula of the form:

$$A_0 \longleftarrow A_1, \dots, A_n. \quad (n \geq 0)$$

where A_0 is either a predication $(p(t_0, \dots, t_n))$, or any description formula except for role inclusion or equality, and A_i ($1 \leq i \leq n$) is a literal, i.e., either a formula like A_0 or its negation. That is, F-Logic rules can include description formulae in their bodies or as their heads. Hence, they can infer description formulae. This is a key feature that enables the dialog between the two reasoners in \mathcal{DFL} . Rules with $n = 0$ are called *facts*. The restrictions on the kinds of description formulae that are allowed, result from the fact that role inclusion and equality are not atoms of F-logic.

The identification of the concept and role semantics with object oriented one leads to expressive integration of rule reasoning with DL reasoning. The smooth embedding of DLs within F-Logic enables seamless combination of rule reasoning with DL reasoning and with reasoning about the DL components as well. Rules can accommodate description formulae, ground or not, as regular atoms, since the inclusion and membership predicates of description logics are understood in the same way in F-Logic. Moreover, since descriptions are terms, rules can include predications over

descriptions. In the following example we demonstrate the expressiveness gained from embedding description reasoning in F-Logic. In the discussion that follows the rules we distinguish between *data level* assertions, which can be either *facts* (general predications) or Abox assertions, to *schema level* assertions, i.e., terminological formulae.

Example 1 F-Logic rules that include description formulae:

1. *Standard data level reasoning – Abox assertions and facts:*

(a) $P \in \text{organic_food_addict} \leftarrow P \in \text{Green_Peace}$

This is the kind of rules supported by BACK, CLASSIC, and formalized in \mathcal{ALCK} .

(b) *A plant that is located near the border is dangerous:*

$P \in \text{dangerous_plant} \leftarrow P \in \text{plant}, (P, L) \in \text{located_at}, L \in \text{border_location}.$

(c) *A plant that is located in a cite in California is also located in the Pacific and near the border:*

$(P, L) \in \mathbf{androle}([\text{located_at_pacific}, \text{located_at_border}]) \leftarrow (P, L) \in \text{located_at_CA}.$ ⁶

(d) *Adding a negated fact to the condition:*

$(P, L) \in \mathbf{androle}([\text{located_at_pacific}, \text{located_at_border}]) \leftarrow (P, L) \in \text{located_at_CA},$
 $\neg \text{private_property}(P)$

(e) *If it is known that all products of a plant are consumed by rich companies then the plant is considered to be a good investment:*

$\text{good_investment}(P) \leftarrow P \in \text{plant}, P \in \mathbf{all}(\text{produces}, \mathbf{some}(\text{consumer}, \text{rich_company})).$

This is the kind of rules supported by \mathcal{AL} -Log and CARIN.

2. *Terminology dependent data level reasoning:*

A plant that is located near the border is dangerous, if border locations are considered risky:

$P \in \text{dangerous_plant} \leftarrow P \in \text{plant}, (P, L) \in \text{located_at}, L \in \text{border_location},$
 $\text{border_location} \leq \text{risky_place}.$

Rules of this kind sanction the applicability of their data level reasoning to knowledge bases with the appropriate terminology (schema).

3. *Terminology (schema) revision rules:*

Border locations are considered risky if peace chances are low and security sensitivity is high:

$\text{border_location} \leq \text{risky_place} \leftarrow \text{peace_chances}(\text{low}), \text{security_sense}(\text{high}).$

4. *Meta-level reasoning about concepts and roles:*

⁶Note that this F-Logic rule is also the role inclusion formula

$$\mathbf{androle}([\text{located_at_pacific}, \text{located_at_border}]) \leq \text{located_at_CA}$$

(a) *If it is known that all product types of a plant are authorized, then the plant is considered to be under control:*

$under_control(P) \leftarrow P \in plant, P \in \mathbf{all}(produces, C), authorized(C).$

Note that C is a concept variable.

(b) *A plant that is located near the border is dangerous, if border locations are considered risky, and provided that the role `located_at` is reliable:*

$P \in dangerous_plant \leftarrow P \in plant, (P, L) \in located_at, L \in border_location,$
 $border_location \leq risky_place, reliable(located_at).$

5. *Knowledge base management rules:*

(a) $contradiction(P, C_2) \leftarrow disjoint(C_1, C_2), P \in C_1, P \in C_1,$
 $greater_than(confidence(C_1), confidence(C_2)).$

(b) $retract(P \in C) \leftarrow contradiction(P, C).$

(c) *A relaxation rule: If C_2 can be considered as a relaxation of C_1 , and the latter is not a strict requirement, then in a contradictory situation replace C_2 for C_1 :*

$assert(P \in C_2) \leftarrow contradiction(P, C_1), retract(P \in C_1), not_strict(C_1),$
 $relaxation_of(C_1, C_2).$

$relaxation_of(\mathbf{atmost}(R, N), \mathbf{atmost}(R, successor(N))).$

6. *Negation as failure versus the classical negation of DLs: Consider the rule “A plant that is not dangerous is safe”. A strict interpretation of this rule is: “A plant that is known to be not dangerous, i.e., a member of the **not**(`dangerouse_plant`) concept, is a safe plant”. This is the classical, monotonic (open world assumption (OWA)) negation. A more credulous interpretation is: “A plant that is not known to be dangerous is safe”. This is the non-monotonic negation, i.e., negation as a failure to prove (closed world assumption (CWA)).*

(a) *Negation under the closed world assumption (CWA):*

$safe_plant(P) \leftarrow P \in plant, \neg(P \in dangerouse_plant).$

(b) *Negation under the open world assumption (OWA):*

$safe_plant(P) \leftarrow P \in plant, P \in \mathbf{not}(dangerouse_plant).$

4 The \mathcal{DFL} Integration Framework

\mathcal{DFL} manages a database of facts and of explicit descriptions in an arbitrary description language. The database has three parts, that include regular factual knowledge, Abox assertions and a terminology:

1. **Part I – Factual knowledge:** Predications $p(t_1, \dots, t_n)$, where p is a predicate symbol that describes a relation, and the t_i -s are any F-Logic terms, ground or not. In particular, the terms can be concept or role descriptions, as in

Kind	No.	Description in words	Formula
Facts	f1)	The temperature of a dangerous plant is high	$temperature(dangerous_plant, high)$
	f2)	Plant pl is controlled by a robot	$controlled_by_robot(pl)$

Table 2: The Facts Part in a \mathcal{DFL} Database.

$relaxation_of(\mathbf{atmost}(R, N), \mathbf{atmost}(R, successor(N))),$
or $authorized(\mathbf{and}(material, ecologically_safe, permit_paid)).$

2. **Part II – Abox:** Extensional, not-necessarily ground, assertions of the form $o \in c$, and $(o_1, o_2) \in r$.
3. **Part III – Terminology (Tbox):** Intensional, not-necessarily ground, definitions and inclusions of concepts and roles.
 - (a) Inclusion formulae: $c_1 \leq c_2, r_1 \leq r_2$.
 - (b) Definition formulae: $c_1 \doteq c_2$, and $r_1 \doteq r_2$.

Table 1 can be extended into a full \mathcal{DFL} database that includes also the factual knowledge in Table 2.

The knowledge base reasons about the given facts and descriptions by consulting two separate reasoners:

1. DL – The *Description Logic* reasoner: A decidable reasoner, that reasons on the basis of the intended meaning of the description constructors that form the descriptions.
2. R – The *Rule* reasoner, that reasons on the basis of given rules and some agreed upon semantical policy (e.g., perfect model for handling negation, or non-monotonic inheritance for handling default inheritance). The rules are F-logic rules that are constructed from atoms that are either description formulae or predications, and are not-necessarily ground. The description atoms are built with the two special predicates for membership (\in) and for inclusion (\leq). They can include non-ground concept and role expressions.

Using Tbox description formulae as atoms in the rules is a unique feature of \mathcal{DFL} . It results from the object-oriented view of DLs as sorted F-Logic languages (see Section 3).

The architecture of \mathcal{DFL} is described in Figure 2. While in query mode the \mathcal{DFL} manager dispatches queries to the two reasoners. The reasoners make efforts to answer. If they succeed, they return an answer(s) to the manager. Intermediate results, obtained by one reasoner, can trigger the other reasoner. This way, a dialog flavored operation is obtained. Note that the two reasoners can operate under different semantical policies, e.g., the OWA for DL and the CWA for R .

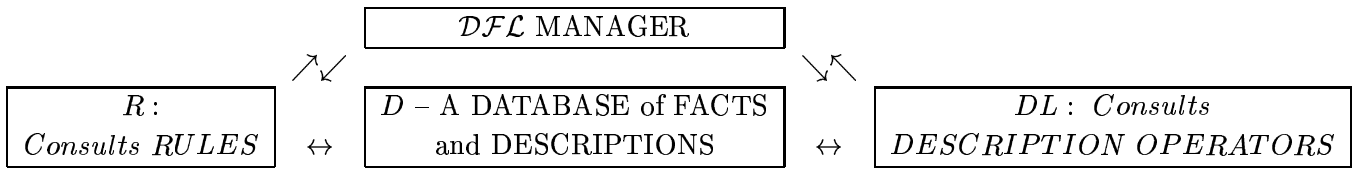


Figure 2: Architecture of a \mathcal{DFL} Knowledge Base

4.1 Semantics of \mathcal{DFL}

Given a \mathcal{DFL} system for a description logic L_P , with a database D , and a rule set $RULES$, it is tempting to define the semantics of \mathcal{DFL} declaratively, as the F-Logic models I such that:

$$I \models D \cup RULES \cup FL_P$$

where FL_P is the corresponding F-Logic theory of L_P . That is, the consequences of a \mathcal{DFL} knowledge base are the formulae that hold in all models of $D \cup RULES \cup FL_P$.

Nevertheless, this idea is wrong since it does not capture the intended operation of \mathcal{DFL} , and it does not lead to a *compositional semantics* of the integrated system. The first problem results from the contra-positive direction of rules, with respect to the classical DL negation constructor **not**. Consider, for example, a \mathcal{DFL} system with a database $o_1 \in \mathbf{not}(d)$, and a rule set $\{X \in d \leftarrow X \in c\}$. By the intended meaning of rules about membership implications (as in CLASSIC, BACK, and \mathcal{ALCK}), $o_1 \in \mathbf{not}(c)$ is not implied. Indeed, following the intended operation of \mathcal{DFL} , as described above, $o_1 \in \mathbf{not}(c)$ is not in the answer set since the DL reasoner cannot infer negated facts (assertions), and the R reasoner does not operate in the contra-positive direction. Yet, it is logically implied according to the suggested declarative semantics. The reason is that the constructor **not** is not the regular rule negation operator. Its meaning is captured by FL_P , and since F-logic is a full-fledged logic, we have:

$$\begin{aligned} & o_1 \in \mathbf{not}(d) \\ \Rightarrow & \neg(o_1 \in d) \\ \Rightarrow & \neg(o_1 \in c) \\ \Rightarrow & o_1 \in \mathbf{not}(c) \end{aligned}$$

The second problem with the suggested declarative semantics is that it is not *compositional*, i.e., the overall semantics of the integrated system is not obtained as a function of the separate semantics of the DL and the R components. Moreover, it implies that the two reasoners operate under the same reasoning policy. Hence, this semantics cannot account for an open (standard) DL component and a non-monotonic rule component. Since most rule components operate under the CWA when handling negation and inheritance, this declarative semantics loses a major feature of a heterogeneous system: The independence of its reasoners. Therefore, the two reasoners in \mathcal{DFL} have each, its own declarative semantics, but no declarative semantics is available as a faithful account for the integration.

4.1.1 Syntactic Compositional Semantics of \mathcal{DFL}

We saw that defining the semantics of \mathcal{DFL} declaratively, on the basis of the underlying F-logic language, does not model the observable behavior of the system, and is not compositional. Compositionality is particularly important for generalizing \mathcal{DFL} to handle multiple information sources, each managing its own information. Consequently, since having a clear semantics is essential for complex systems, the denotation of \mathcal{DFL} is defined operationally, by syntactic objects, based on the independent denotations of the DL and R reasoners. This way we obtain *compositional semantics*, that allows the reasoners to operate along different reasoning policies, and does not break the uni-directional nature of the rule reasoning.

More concretely, the syntactic denotation of a \mathcal{DFL} knowledge base consists of a set of description formulae (ground or not). It assumes that the DL and the R reasoners are associated with *denotation operators* \mathcal{DL}^P and \mathcal{R}^{RULES} , respectively, that are mappings over the power set of the syntactic objects that make up the semantics. It is constructed by *iterative application* of \mathcal{DL}^P and \mathcal{R}^{RULES} to D . This way the principles of *modularity* and *compositionality* are kept, and the observable behavior of a rule system is correctly modeled.

The semantics of \mathcal{DFL} , denoted $\mathcal{DFL}_{P,RULES}(D)$ is defined as follows:

$$\begin{aligned}
 \text{Define:} \quad & T_{P,RULES}(D) \stackrel{def}{=} \mathcal{DL}^P(D) \cup \mathcal{R}^{RULES}(D) \\
 \text{and} \quad & T_{P,RULES}^0(D) = D \\
 & T_{P,RULES}^{k+1}(D) = T_{P,RULES}(T_{P,RULES}^k(D)) \quad k \geq 0 \\
 & T_{P,RULES}^\omega(D) = \bigcup_{k=0}^{\infty} T_{P,RULES}^k(D) \\
 \text{Then:} \quad & \mathcal{DFL}_{P,RULES}(D) \stackrel{def}{=} T_{P,RULES}^\omega(D)
 \end{aligned}$$

The compositional semantics does not specify the \mathcal{DL}^P and \mathcal{R}^{RULES} mappings, which express the intended meanings of the components. We define them declaratively, using the set-theoretic semantics of description logics for the DL component, and the model-theoretic semantics of F-Logic for the R component. For a rule component that includes only pure logic programming rules, we also provide a fixpoint semantics, which helps in designing an inference procedure.

Notation: For a set P of description constructors, let L_P denote the description language over P , and $varL_P$ denote the non-ground extension of L_P . That is, $varL_P$ includes also L_P formulae with variables replaced for individuals, concepts or roles. Note that the role inclusion and equality formulae in L_P and $varL_P$ are not atoms but rules in F-Logic (see Section 3). For a \mathcal{DFL} database D , let $D_{/varL_P}$ denote the restriction of D to possibly non-ground description formulae, and $D_{/L_P}$ denote the set of all consistent instantiations of $D_{/varL_P}$ formulae over the constant symbols in D (a consistent instantiation is sensitive to sort). For a set of rules $RULES$, let $RULES_{/varL_P}$ denote the restriction of $RULES$ to the rules whose condition (body) atoms are description formulae in $varL_P$ (no predications in the body). Then,

Definition 4.1 The \mathcal{DL}^P and the \mathcal{R}^{RULES} denotational mappings:

$$1. \mathcal{DL}^P(D) = \{\gamma \mid D_{L_P} \stackrel{DL}{\models} \gamma\}$$

2. $\mathcal{R}^{RULES}(D) = \{\gamma \mid D \cup RULES \stackrel{FL}{\models} \gamma, \gamma \text{ is an F-Logic rule whose condition atoms are all in } varL_P\}$

The rationale behind the definition of $\mathcal{R}^{RULES}(D)$ is that the atoms in $varL_P$ should function as *open atoms*, i.e., atoms that are partially defined in the rules. Therefore, in order to keep the semantics compositional, $\mathcal{R}^{RULES}(D)$ cannot be defined as a set of atoms (see the example for the non-compositionality of \mathcal{AL} -log, CARIN, and \mathcal{ALCK} , in Subsection 2.2). Note also that $\mathcal{DL}^P(D)$ includes only (ground) description formulae (formulae from L_P).

4.1.2 Properties of the Syntactic Compositional Semantics

1. Source inclusion in the component semantics:

- (a) $\mathcal{R}^{RULES}(D) \supseteq D \cup RULES_{/varL_P}$
- (b) $\mathcal{DL}^P(D) \supseteq D_{/L_P}$

2. Compositional faithfulness – The compositional semantics includes all the consequences of the individual components:

- (a) If $D_{L_P} \stackrel{DL}{\models} \gamma$ then $\gamma \in \mathcal{DFL}_{P,RULES}(D)$.
- (b) If γ is an F-Logic rule whose condition atoms are all in $varL_P$ and $D \cup RULES \stackrel{FL}{\models} \gamma$ then $\gamma \in \mathcal{DFL}_{P,RULES}(D)$.

3. Soundness and inconsistency – The compositional framework is not immune against inconsistencies. In fact, it is quite likely that if the subject domains of autonomous information sources overlap, their inferences might contradict. Resolving inconsistencies is a major subject in the area of Hybrid Knowledge Bases (see for example, [40, 57]). For DL reasoners, consistency checkers such as CRACK ([20]) can be used. In any case, handling inconsistencies is outside the scope of this paper, and should be resolved on a system dependent basis.

If the components are not contradictory, then the syntactic compositional semantics is logically implied from their union. That is:

Claim 4.2 *If $D \cup RULES \cup FL_P$ is satisfiable, then $D \cup RULES \cup FL_P \stackrel{FL}{\models} \mathcal{DFL}_{P,RULES}(D)$.*

Proof: Induction on the iterative $\mathcal{DFL}_{P,RULES}(D)$ construction. \square

4. Filtered consequences – The compositional semantics may sanction consequences of the union of the components. That is,

$$D \cup RULES \cup FL_P \stackrel{FL}{\models} \gamma \not\Rightarrow \gamma \in \mathcal{DFL}_{P,RULES}(D).$$

This property is desirable for consequences that result from the contra positive direction of rules. But it also restricts the expressive power of the \mathcal{DFL} framework. In particular, \mathcal{DFL} does not support case analysis on rule conditions, since rule semantics requires full instantiation of their condition part.

5. **The declarative definition of $T_{P,RULES}$** – If the \mathcal{R}^{RULES} mapping is not monotonic, the application order of \mathcal{R}^{RULES} and \mathcal{DL}^P might affect the resulting inferences. Consequently, we deliberately defined the operator $T_{P,RULES}$ as the union of the denotation mappings. Again, contradicting beliefs should be resolved on a system dependent basis.

4.1.3 The unfolding fixpoint semantics for a rule component with pure logic programming rules:

The standard immediate consequences operator in logic programming computes ground atoms of the language ([30]). The fixpoint semantics defined on the basis of this operator consists of ground facts, membership formulae and concept description formulae. The *s-semantics* approach of [16] extends the fixpoint semantics to include non ground atoms, but still cannot capture the intended \mathcal{R}^{RULES} mapping. The following example shows the limited expressiveness of a fixpoint semantics of atoms.

Example 2 Consider the rules:

$$\begin{aligned} (X, Y) \in r_2 &\leftarrow p. \\ p &\leftarrow (X, Y) \in r_1. \end{aligned}$$

The rule $(X, Y) \in r_2 \leftarrow (X, Y) \in r_1$ (called their *resolvent*), which is the role inclusion $r_1 \leq r_2$, is certainly logically implied. This formula can trigger further inferences by the *DL* reasoner, at the next stage of the iteration. Yet, it is not computed by a fixpoint operator since it is not an atom. Of course, we can restrict the \mathcal{R}^{RULES} to atoms alone, but doing so blocks desirable interaction that involves role inclusion formulae (see also [10]).

This problem reminds the case of open logic programs, where the standard fixpoint semantics of atoms blocks the computation of predicate symbols that are partially defined. Consider the following example, taken from [16]:

Example 3 Given the two programs

$$P = \left. \begin{array}{l} \{ r(b) \leftarrow p(b). \\ p(a). \end{array} \right\} \quad Q = \{ p(b). \}$$

The fixpoint semantics of P and Q , are $\{p(a)\}$ and $\{p(b)\}$, respectively, while the fixpoint semantics of $P \cup Q$ is $\{p(a), p(b), r(b)\}$. Therefore, the fixpoint semantics is not compositional, i.e., the semantics of $P \cup Q$ cannot be computed from those of P and Q .

In the case of open logic programs the semantical problem is corrected by replacing the fixpoint semantics of atoms by a semantics of *resultants* that allows to delay the evaluation of open atoms. The open semantics is defined as the least fixpoint of an *unfolding operator* that computes resolvents of the rules.

In our case, the fixpoint semantics of atoms that limits the combined expressiveness, is replaced by an unfolding semantics of resultants that allows to delay the evaluation of atoms in $varLP$. That is, the \mathcal{R}^{RULES} mapping can be defined as the least fixpoint of an *unfolding operator* $unf_P(Q)$

which, intuitively, computes resolvents of rules in P and Q (for a precise definition consult ([16])). The unfolding is designed to compute rules whose condition atoms are open. We use the following notation: FL_{varLP} denotes the set of F-Logic rules whose condition part includes only $varLP$ atoms, and Id_{LP} is the set of rules $\{dl \leftarrow dl \mid dl \text{ is a non-ground membership or inclusion atom}\}$ (i.e., dl is $X \in C$, or $(X, Y) \in R$ or $C \leq D$).

Definition 4.3 The unfolding semantics for the rule component: Let $R \subseteq FL_{varLP}$. Define an operator:

$$T_D^{RULES}(R) = unf_{D \cup RULES}(R \cup Id_{LP})$$

Based on the results in [16], we have:

Claim 4.4 T_D^{RULES} is continuous with respect to the subset order relation, and $\mathcal{R}^{RULES}(D) = T_D^{RULES} \uparrow \omega$.

The unfolding fixpoint semantics for the rule component implies a constructive way to compute $\mathcal{R}^{RULES}(D)$. Example 4 computes the compositional semantics $\mathcal{DFL}_{P, RULES}(D)$ for two example databases and rule sets.

Example 4

In the following two examples, role relations that are obtained by the \mathcal{R}^{RULES} mapping, allow the DL reasoner to obtain new descriptions, that could not be obtained otherwise.

1. P : {some}
 D : $c \leq \mathbf{some}(r_1, d)$
 $RULES$: $(X, Y) \in r_2 \leftarrow p$
 $p \leftarrow (X, Y) \in r_1$.
 $\mathcal{DFL}_{P, RULES}(D) = D \cup RULES \cup \{r_1 \leq r_2, c \leq \mathbf{some}(r_2, d),$
 $\mathbf{some}(r_1, c) \leq \mathbf{some}(r_2, c), \mathbf{some}(r_1, d) \leq \mathbf{some}(r_2, d) \dots\}$

For this example, taking the semantics of the rule component as the fixpoint of the standard immediate consequent operator, yields a strictly weaker semantics:

$$\{c \leq \mathbf{some}(r_1, d)\} \subset \mathcal{DFL}_{P, RULES}(D)$$

2. $P =$ {and-role, compose, some}
 D : $r \leq \mathbf{and-role}(\mathbf{compose}(r, r), r)$
 $a \in \mathbf{some}(r, c)$
 $RULES$: $(X, Y) \in \mathbf{compose}(\mathbf{compose}(R, R), \mathbf{compose}(R, R)) \leftarrow$
 $(X, Y) \in \mathbf{compose}(R, R)$
 $\mathcal{DFL}_{P, RULES}(D) = D \cup RULES \cup \{r \leq \mathbf{compose}^i(r, r) \mid i \geq 1\}$
 $\cup \{a \in \mathbf{some}(\mathbf{compose}^i(r, r), c) \mid i \geq 1\}$
where $\mathbf{compose}^1(r, r) = \mathbf{compose}(r, r),$
 $\mathbf{compose}^{i+1}(r, r) = \mathbf{compose}(\mathbf{compose}^i(r, r), \mathbf{compose}^i(r, r))$

As in the previous example, the descriptions in $\{r \leq \mathbf{compose}^i(r, r) \mid i \geq 1\}$ and in $\{a \in \mathbf{some}(\mathbf{compose}^i(r, r), c) \mid i \geq 1\}$ are not included in a semantics based on the standard immediate consequent operator of atoms.

4.2 Reasoning in \mathcal{DFL}

The compositional semantics of \mathcal{DFL} sets its own soundness and completeness criteria. A standard reasoning task starts with a goal to prove or compute; a sound and complete inference procedure needs to compute exactly the elements of $\mathcal{DFL}_{P,RULES}(D)$. The inference is affected by two factors:

- The inference procedures of the components, termed henceforth *interpreters*.
- The integration procedure. Achieving the compositional semantics requires some form of dialog between the reasoners.

Clearly, complete reasoning behavior can be obtained by simulating the iterative definition of the compositional semantics, i.e., with complete interpreters that repeatedly write their inferences in the database (similarly to the *naive* and the *semi-naive* algorithms for computing the transitive closure of a relation in databases). However, this procedure is not practical, since it is not focused on the given query. A more focussed behavior can be obtained with goal directed interpreters that dispatch queries to each other. But this approach requires decisions as to *what* and *when* to dispatch a query.

A combination of these two kinds of dialogs yields the *make-failure-public* dialog policy, that uses goal directed interpreters that have the following properties:

1. Each interpreter writes its failures in the database.
2. Each interpreter can resume previously failed proofs.

Failure writing acts like query dispatch: Once an interpreter succeeds in proving a failure of the other interpreter, the latter might complete its computation successfully. Sharing failures with the other interpreter guarantees that the dialog is not stuck with one interpreter, and saves the need for timing the query dispatch. If the reasoners are complete with respect to their separate semantics, the combined approach yields a complete implementation.

The requirements that the *make-failure-public* policy set on the interpreters are met by magic-set-with-tabling interpreters in logic programming. The magic-set approach simulates the control of a goal directed proof ([58, 59, 63]). Adding tabling yields the desired properties for the two reasoners, since a magic-set-with-tabling interpreter is complete and task-focused. It documents failures in the form of goals that are called but never succeed, and can continue failed proof lines, once failed goals are proved by another interpreter. That is, the tabling acts as a communication channel between the reasoners: The tabling of failures by one interpreter can trigger the other one.

Consequently, we suggest to base reasoning in \mathcal{DFL} on reasoners that act like magic-set-with-tabling interpreters. Tabling based approaches usually trade time for space. In certain cases they replace infinite proof branches for finite proofs, but the price is a much larger database. The size of the database can be reduced by restricting the tabling to description formulae alone, and generalizing the tabled goals. Likewise, syntactic invariants (goals that are equal up to variable renaming) should not be duplicated, and DL tautologies should not be tabled. Unfortunately, the tabling cannot be further generalized to exclude goals that subsume each other (in the sense of term subsumption in logic), since a *DL* reasoner operates on ground goals (once a non-ground goal like $call(c \leq \mathbf{some}(R, d))$ is tabled, it blocks the tabling of more concrete goals).

4.2.1 DL Reasoning within \mathcal{DFL}

As described in Subsection 2.1.3, the two main approaches in DL reasoning are the *constraint based (tableaux) refutation approach* (KRIS, CRACK, FaCT), and the *structural subsumption approach* (BACK and CLASSIC). Using constraint DL reasoning in the make-failure-public dialog policy is hard, since the constraints approach is not reductional. In order to report failures in proving subgoals that are needed to complete proofs, a system needs to workout some reductions of its goals. But the constraints approach works by forward propagation of given information until contradictions, or canonical models are constructed.

The weaker structural subsumption approach fits well into the dialog based integration in \mathcal{DFL} , since it is reductional. A structural inference method can be formulated as a set of structural rules, for reducing the given subsumption query into simpler subsumption queries, based on the meaning of the constructors. For example, the following rules are based on [48] (they are written in a Prolog style):

- 1 $nothing \leq - .$
- 2 $T \leq T : - primitive_symbol(T).$
- 3 $C1 \leq \mathbf{and}([C2]) : - C1 \leq C2.$
- 4 $C1 \leq \mathbf{and}([C2|T]) : - C1 \leq C2, C1 \leq \mathbf{and}(T).$
- 5 $\mathbf{and}([C1|T]) \leq C2 : - C1 \leq C2.$
- 6 $\mathbf{and}([_|T]) \leq C2 : - \mathbf{and}(T) \leq C2.$
- 7 $\mathbf{all}(Rq, Cq) \leq \mathbf{all}(Rp, Cp) : - Cq \leq Cp, Rp \leq Rq.$
- 8 $\mathbf{atmost}(0, Rq) \leq \mathbf{all}(Rp, Cp) : - Rp \leq Rq.$
- 9 $\mathbf{atleast}(Nq, Rq) \leq \mathbf{atleast}(Np, Rp) : - Rq \leq Rp, Np \leq Nq.$
- 10 $\mathbf{atmost}(Nq, Rq) \leq \mathbf{atmost}(Np, Rp) : - Rp \leq Rq, Nq \leq Np.$

The \mathcal{DFL} system that implements the \mathcal{DFL} framework with the make-failure-public integration policy is described in Appendix A. Appendix B demonstrates a \mathcal{DFL} knowledge base and query answering, that requires an intensive dialog between the two reasoners. The rules in this knowledge base include intensional description formulae, and the description formulae appear both in the bodies and in the heads of rules. Such rules extend the kinds supported by BACK, CLASSIC, CARIN, \mathcal{AL} -log, and \mathcal{ALCK} .

\mathcal{DFL} Evaluation: The comparison of \mathcal{DFL} with other DL-Rule integrations has several aspects. Clearly, the rule language of \mathcal{DFL} is far more expressive than in any of the known integrations, since it can express terminology formulae and apply predications to description expressions. In particular, the schema (terminology) can be used to condition rule applications, rules can be used for schema revision, the rules allow for meta level reasoning about concepts and roles, and can even support database management. In addition, the \mathcal{DFL} semantics is compositional, while all others are not. Nevertheless, on the disadvantages side, the semantics of \mathcal{DFL} is operational, while CARIN, \mathcal{AL} -Log and \mathcal{ALCK} enjoy a declarative semantics. Moreover, \mathcal{DFL} does not support case analysis on rule conditions, since rule application requires full instantiation of their condition part.

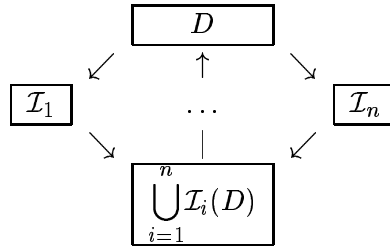


Figure 3: Structure of the compositional semantics of \mathcal{H}

5 Generalizing the $DF\mathcal{L}$ Architecture to Integration of Multiple Reasoners Over a Common Domain

The $DF\mathcal{L}$ architecture can be naturally generalized to heterogeneous systems that manipulate multiple information sources with no declarative semantics for the integration. The integration of the different sources is done via a global database accessible to all sources. The inferences made by an information source are accessible to all other sources. The semantics of the overall system is defined as a *Composition* of the semantics of the integrated sources, and hence preserves their *Modularity*. The operation of the system is judged against its syntactic compositional semantics.

Let \mathcal{H} be a heterogeneous system with information sources I_1, I_2, \dots, I_n , and a global database D . An information source I_i , ($1 \leq i \leq n$) is associated with a set of syntactic objects D_i , that characterizes the subject domain of I_i , and with a semantic operator \mathcal{I}_i , that describes the meaning of I_i as a subset of D_i . That is, \mathcal{I}_i maps subsets of $\bigcup_{i=1}^n D_i$ into subsets of D_i . D is a subset of $\bigcup_{i=1}^n D_i$. The compositional semantics, denoted $\mathcal{H}_{I_1, \dots, I_n}(D)$, is constructed by *iterative application* of the \mathcal{I}_i operators ($1 \leq i \leq n$) to D . This way the principles of *modularity* and *Compositionality* are kept. \mathcal{H} is formally defined as follows:

Define: $T_{I_1, \dots, I_n}(D) \stackrel{def}{=} \bigcup_{i=1}^n \mathcal{I}_i(D)$

Let $T_{I_1, \dots, I_n}^0(D)$ be a system dependent initial version of D (denoted $D_{I_1, \dots, I_n}^0(D)$).

$$T_{I_1, \dots, I_n}^{k+1}(D) = T_{I_1, \dots, I_n}^{\infty}(T_{I_1, \dots, I_n}^k(D)) \quad k \geq 0$$

$$T_{I_1, \dots, I_n}^{\omega}(D) = \bigcup_{k=0}^{\infty} T_{I_1, \dots, I_n}^k(D)$$

Then: $\mathcal{H}_{I_1, \dots, I_n}(D) \stackrel{def}{=} T_{I_1, \dots, I_n}^{\omega}(D)$

The general structure of the compositional semantics is visualized in Figure 3.

Property: In the general case, $\bigcup_{i=1}^n \mathcal{I}_i(D) \subset \mathcal{H}_{I_1, \dots, I_n}(D)$. That is, the interaction between the information sources creates new information.

Achieving a Task-Focused Complete Implementation

Following the discussion in Section 4.2, we distinguish between three kinds of dialogs:

1. The *make-public based dialog*, in which the information sources repeatedly write their inferences in the global database. This kind is unfocused, since the information made public by one party may be irrelevant to other parties.
2. The *dispatch based dialog*, in which a party dispatches a query only when it needs it for its future functioning. This kind is goal directed, but requires decisions as to *what* and *when* to ask.
3. The combined approach, in which parties make public their failures (and possibly some of their successes). Failures that are made public act like query dispatch: Once a party succeeds in proving a failure of another party, the latter party might complete its computation successfully. Sharing failures with all other parties guarantees that the dialog is not stuck with one party, and saves the need for timing the dispatch. Therefore, the combined dialog is both focused and simple to design.

Using the combined dialog kind with the following system behavior and interpreter properties lead to a complete, focussed implementation:

1. System behavior:

- The interpreters operate concurrently.
- The initial query is first checked in the global database (or rather in its initial version, $D_{I_1, \dots, I_n}^0(D)$). If it is not already there, it triggers all the interpreters. A failure of an interpreter that is announced in the global database, triggers all other interpreters.

2. Interpreter properties:

- The interpreters used by the reasoners are goal directed, and complete with respect to the semantics of their reasoners.
- An interpreter documents in the global database every failure that it encounters in its efforts to compute its goals.
- An interpreter can resume a previously failed proof, once another interpreter succeeds in proving the failure that caused the proof to stop. This feature can be achieved by tabling of all intermediate results. If the tabled results are made public, other interpreters might benefit from them as well.

This claim characterizes the external properties of interpreters that implement reasoners in a heterogeneous system. The desired features are:

- Complete and goal directed interpreters.
- Interpreters document their failures.
- Capability to resume previously failed proofs.

6 Conclusion and Future Research

In this paper we introduced a formal scheme for the integration of information sources for which a combined declarative semantics is not available. The integration is defined by a syntactic compositional semantics. That is, the overall meaning of the integrated system is defined as a combination of the independent meanings of its components. We also suggested an architecture and a *make failure public* computation process that can achieve a complete focussed performance.

This scheme was used to formalize the integration of a description logic reasoner and an expressive rule reasoner, for which a combined declarative semantics is not known. To the best of our knowledge, the *DFL* system is the first to support a true dialog between a DL and a rule reasoners, that operate under different semantical policies – OWA for the DL reasoner, and CWA for the Rule reasoner. The dialog is enriched by the F-Logic view of DLs, that enables terminology conditioned factual reasoning, schema revision rules, and meta-level reasoning about a terminology.

The *DFL* system can be extended to integrate a variety of reasoners, including multiple DL reasoners, CARIN or *AL*-log like reasoners, several rule reasoners, and reasoners of other types. Given interpreters that meet the requirements of the make-failure-public policy, the *DFL* architecture can support their integration into a broad heterogeneous system. The cooperative dialog approach is already in use in a system that supports cooperative query answering in federated databases ([14, 11]).

References

- [1] M. Aiello, C. Areces, and M. de Rijke. Spatial reasoning for image retrieval. In *International Workshop on Description Logics*, pages 23–27, Linkoping, Sweden, 1999.
- [2] A. Artale and E. Franconi. A computational account for a description logic of time and action. In *KR-94*, pages 3–14, 1994.
- [3] A. Artale and E. Franconi. Reasoning with enhanced temporal entity-relationship models. In *Proceedings of the 1999 Entity-Relationship Conference ER-99*, pages 81–95, 1999.
- [4] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *IJCAI-91*, 1991.
- [5] F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *IJCAI-91*, pages 452–457, 1991.
- [6] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proceedings of the Workshop on Processing Declarative Knowledge, PDK-91, Lecture Notes in AI*, pages 67–86. Springer-Verlag, 1991.
- [7] F. Baader and B. Hollunder. How to prefer more specific defaults in terminological default logic. In *IJCAI-93*, pages 669–674, 1993.

- [8] F. Baader, B. Hollunder, B. Nebel, H. Profitlich, and E. Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *KR-92*, pages 270–281, 1992.
- [9] M. Balaban. The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, 15:19–60, 1995.
- [10] M. Balaban. Compositional semantics for descriptions knowledge bases with rules. Technical report, Information Systems Program, Department of industrial Engineering and Management, Ben-Gurion University, Beer Sheva, Israel, 1996. ftp ftp.cs.bgu.ac.il, cd pub/people/mira.
- [11] M. Balaban, N. Berezansky, and E. Gudes. Answering cooperative recursive queries in federated databases. Technical report, Department of Computer Science, Ben-Gurion University, Beer Sheva, Israel, 2001.
- [12] C. Beery, A. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS-97)*, pages 99–108, 1997.
- [13] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Terminological logics for schema design and query processing in oodbs. In *Proceedings, KI-94 Workshop on Reasoning about Structured Objects – Knowledge Representation Meets Databases*, Saarbrucken, Germany, 1994.
- [14] N. Berezansky. *Cooperating Queries in Federated Databases*. M.Sc. thesis, Department of Computer Science, Ben-Gurion University of the Negev, ISRAEL, 2001.
- [15] A. Borgida, R. Brachman, D. McGuinness, and L. Resnick. Classic: A structural data model for objects. In *ACM-SIGMOD-89*, Portland, OR, 1989.
- [16] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *J. of Logic Programming*, 12(3):187–230, 1993.
- [17] R. Brachman, V. Gilbert, and H. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts of krypton. In *IJCAI-85*, pages 532–539, 1985.
- [18] R. Brachman and H. Levesque. Competence in knowledge representation. In *AAAI-82*, pages 189–192, Pittsburgh, PA, 1982.
- [19] R. Brachman and J. Schmolze. An overview of the kl-one knowledge representation system. *Cognitive Science*, 9:171–216, 1985.
- [20] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: a preliminary report. In *International Workshop on Description Logics*, pages 131–139, Roma, Italy, 1995.
- [21] M. Buchheit, F. Donini, W. Nutt, and A. Schaerf. Terminological systems revisited: Terminology = schema + views. In *Proceedings, AAAI-94*, pages 199–204, 1994.

- [22] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Answering queries using views in description logics. In *International Workshop on Description Logics*, pages 9–13, Linköping, Sweden, 1999.
- [23] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class-based representation formalisms. In *KR-94*, pages 109–120, 1994.
- [24] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.
- [25] E. Compatangelo, F. Donini, and G. Rumolo. A description logic for reasoning with behavioral knowledge. In *International Workshop on Description Logics*, pages 44–48, Gif-sur-Yvette, France, 1997.
- [26] E. Domenicucci, F. Donini, and M. Schaerf. Icarus: Intelligent classification and retrieval of unlabelled scenes. In *International Workshop on Description Logics*, pages 46–50, Linköping, Sweden, 1999.
- [27] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. A hybrid system with datalog and concept languages. In *Trends in AI, volume LNAI 549*. Springer Verlag, 1991.
- [28] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge representation, Studies in Logic, Languages, and information*, pages 193–238, 1996.
- [29] F. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. An epistemic operator for description logics. *AIJ*, 100:225–274, 1998.
- [30] M. Emden and R. Kowalski. The semantics of predicate logic as a programming language. *J. of the ACM*, 23(4):733–742, 1976.
- [31] A. Eyal. *D $\mathcal{F}\mathcal{L}$ – A Truly Heterogeneous System with a Description Logic and a Rule Components*. M.Sc. thesis, Department of Mathematics and Computer Science, Ben-Gurion University of the Negev, ISRAEL, 1998.
- [32] P. Gonzales-Calero, B. Diaz-Agudo, and M. Gomez-Albarran. Applying dls for retrieval in cased-based reasoning. In *International Workshop on Description Logics*, pages 61–55, Linköping, Sweden, 1999.
- [33] V. Haarslev and R. Moeller. Spatioterminological reasoning: Subsumption based on geometrical inferences. In *Proceedings of the Workshop on Description Logics*, pages 74–78, 1997.
- [34] R. Himmeroder and C. Schlepphorst. Florid version 1.0: User manual. Technical report, Institut für Informatik, Universität Freiburg, Freiburg, Germany, July 1996.
- [35] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *ECAI-90*, pages 348–353, 1990.

- [36] I. Horrocks. Using an expressive description logic: Fact or fiction? In *KR-98*, pages 636–647, 1998.
- [37] I. Horrocks, A. Rector, and C. Goble. A description logic based schema for the classification of medical data. In *ECAI-96 Workshop on Knowledge Representation Meets Databases*, pages 24–28, Dudapest, Hungary, 1996.
- [38] M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *SIGMOD-89*, 1989.
- [39] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.
- [40] M. Kifer and V. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12(4):335–368, 1992.
- [41] H. Levesque and R. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.
- [42] A. Levy and M.-C. Rousset. Carin: A representation language combining horn rules and description logics. In *KR-96*, pages 323–327, 1996.
- [43] A. Levy and M.-C. Rousset. The limits on combining recursive horn rules and description logics. In *AAAI-96*, 1996.
- [44] K. Luck, B. Nebel, C. Peltason, and A. Schmiesel. The back system. Technical Report KIT Report 29, Department of Computer Science, Technische Universität Berlin, Berlin, FRG, 1985.
- [45] K. Luck, B. Nebel, C. Peltason, and A. Schmiesel. The anatomy of the back system. Technical Report KIT Report 41, Department of Computer Science, Technische Universität Berlin, Berlin, FRG, 1987.
- [46] R. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 385–400. Morgan Kaufmann, 1991.
- [47] B. Nebel. Computational complexity of terminological reasoning in back. *J. of Artificial Intelligence*, 34:371–383, 1988.
- [48] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Dissertation, University of Saarlands, Saarbrücken, 1989.
- [49] B. Nebel. Terminological reasoning is inherently intractable. *J. of Artificial Intelligence*, 43:235–249, 1990.
- [50] P. Patel-Schneider. Undecidability of subsumption in nkl. *J. of Artificial Intelligence*, 39:263–272, 1989.

- [51] J. Quantz and V. Royer. A preference semantics for defaults in terminological logics. In *KR-92*, pages 294–305, 1992.
- [52] L. Resnick, A. Borgida, R. Brachman, D. McGuinness, and P. Patel-Schneider. Classic description and reference manual for common lisp implementation. Technical Report Version 1.02, AT&T Bell Labs, 1990.
- [53] R. Rosati. Towards expressive kr systems integrating datalog and description logics: Preliminary report. In *International Workshop on Description Logics*, pages 160–164, Linköping, Sweden, 1999.
- [54] M. Schmidt-Schauß. Subsumption in kl-one is undecidable. In *Proceedings, Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, Toronto, Ontario, Canada, 1989.
- [55] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *J. of Artificial Intelligence*, 48(1):1–26, 1991.
- [56] M. Stickel. A non-clausal connection-graph resolution theorem proving program. In *AAAI-82*, pages 229–233, 1982.
- [57] V. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.
- [58] H. Tamaki and T. Sato. Old resolution with tabulation. In *3rd International Conference on Logic Programming*, pages 84–98, 1986.
- [59] J. Ullman. *Principles of Database and Knowledge-base Systems*. Computer Science Press, 1989.
- [60] J. Widom and S. Ceri. *Active Database Systems*. Morgan-Kaufmann, 1996.
- [61] W. Woods. Understanding subsumption and taxonomy: A framework for progress. In J. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 45–94. Morgan Kaufmann, 1991.
- [62] W. Woods and J. Schmolze. The kl-one family. *Computers and Mathematics with Applications*, Special Issue on Semantic Networks in Artificial Intelligence, 1992.
- [63] J. Wunderwald. Memoing evaluation by source-to-source transformation. In *LOPSTR-95*, 1995.

A The \mathcal{DFL} Implemented System

The \mathcal{DFL} system implements the make-failure-public approach. It cannot use existing DL systems like BACK and CLASSIC, or the F-Logic system FLORID (F-Logic Reasoning In Databases [34]), that were built as stand-alone, encapsulated systems, and do not fit into a plug-in system of components.

The DL and the R reasoners in the \mathcal{DFL} system are implemented as magic-set-with-tabling interpreters. System operation is triggered by a conjunctive goal of description formulae and predications (not-necessarily ground). The system works out the goal as a Prolog goal, from left to right. Both interpreters try to prove the goal, and use the database for tabling. In order to keep the size of the database small, the interpreters perform controlled tabling. Only calls and answers to descriptive goals are tabled, since only description formulae can trigger a dialog between the two reasoners. The system operates by repeated applications of the two interpreters. In each step, an interpreter is triggered by failures (calls without answers) and successes (answers to goals) of the other interpreter. The overall operation is improved by using particularly efficient interpreters.

Structural Subsumption in the DL Component of \mathcal{DFL}

Structural subsumption based systems usually operate under the *expand-normalize-compare* method. That is, given a goal to prove subsumption between concepts, or instance of a concept, the system first performs full expansion on the concept descriptions in the goal, based on the definitions in the terminology. This step results concept descriptions that include only undefined concept symbols (primitives). The full expansion is possible since the terminology is acyclic. The resulting concept descriptions are then normalized, using equivalence preserving transformations, and compared.

In the context of the dialog based integration system, expansion is not feasible since it can ruin the dialog with the other reasoner, and can explode the rule basis (expansion can yield concept descriptions that are exponential in the size of the terminology [49]). Therefore, the \mathcal{DFL} algorithm does not perform *expand* and *normalize* on the database. Instead, it takes an *expand-by-demand* approach. That is, if no structural rule applies, the algorithm performs a single expand and tries to prove the goal again. The expand is achieved by a rule of inclusion transitivity as follows:

$$X \leq Y : -X \leq Z, Z \leq Y.$$

Subsumption proofs can terminate by unifying possibly large portions of the compared descriptions. This is achieved by the fact

$$T \leq T.$$

The expand-by-demand structural subsumption achieves meaningful savings in symbol expansions, though at a small lose of expressive power (for details, consult [31]).

Claim A.1 *If the terminology includes no definitional cycles, and no cycles are generated by the R reasoner, then under a tabling policy the DL interpreter is guaranteed to terminate.*

Proof: Replacing a definition $a = def$ by two inclusions $a \leq def$ and $def \leq a$ creates an immediate inclusions cycle. However, sub-expressions of def cannot expand to include a . Hence, cycles can

be generated only by infinite repetitive replacements of a with def and vice-versa. Such chains cannot expand for ever since, apart from the transitivity rule, all other rules shorten the tested descriptions. Consequently, there is a point in a proof chain where only the transitivity rule applies. At that point, def can be replaced by a and tabling cuts this proof branch. \square

Note that since the DFL algorithm performs only step-by-step expansion, it can apply to cyclic terminologies, but this can lead to infinite proof branches.

Rule unfolding in the Rule Component of DFL

The rules reasoner is implemented by a magic-set-with-tabling interpreter that is extended with “selective” unfolding on the rules, that computes role inclusion formulae, which are rules and not atoms in F-Logic. When the R component infers, through unfolding, a rule of the form $(X, Y) \in role_expression_1 : - (X, Y) \in role_expression_2$, it writes in the database the role inclusion formula $role_expression_1 \leq role_expression_2$. The Rule reasoner performs unfolding only on rules whose head can be the head of a role inclusion formula, i.e., $(X, Y) \in role_expression$.

Since the rules of R are F-Logic rules, the interpreter for R needs also to implement the relevant parts of F-Logic. So far we use only the inclusion and membership relations of F-Logic. We did not deal with F-Logic inheritance, negation, or types.

B A DFL Knowledge Base and Query Answering

Example 5 A DFL knowledge base:

Set of description constructors:

{ all , atleast , atmost , and (for concept and role) }

Notation: $::$ stands for membership.

$<=$ stands for concept inclusion.

$<*$ stands for role inclusion.

$<=>$ stands for concept equality.

Rules:

L is a risky_place -if- there is a dangerous_plant P that produces (waste) X which is buried_at L, and this plant is not under government control.

```
r1) L :: risky_place :-
    P :: dangerous_plant,
    (P,X) :: produces,
    (X,L) :: buried_at,
```


P is a candidate_for_closing -if- it is:
a dangerous_plant type,
and a dangerous_plant is explodable,
and all the known products that P produces are toxic_waste,
and plants that produce only toxic_waste do not always bury their products
in a safe place.

r8) candidate_for_closing(P):-
P :: dangerous_plant,
dangerous_plant <= explodable,
all_known(P, produces, toxic_waste),
\+ all(produces,toxic_waste) <= all(buried_at,safe_place).

r9) temperature(dangerous_plant,60).

r10) no_gov_control(plant_Machteshim).

All known R-s of X are C-s:

r11) all_known(X,R,C):-
findall(Y, (X,Y)::R ,List),
ins_all(List, C).

All elements of a list are instances of C:

r12) ins_all([],_).
ins_all([H|T],C) :- H::C, ins_all(T,C).

Data Base:

d1 (plant_Machteshim,waste_NaCl) :: produces.
d2 plant_Machteshim :: chemical_plant.
d3 chemical_plant <=>
and([plant, all(produces,chemical_product)]).
d4 waste_Br2 :: and([waste,chemical_material]).
d5 ramatHovav_ind :: near_population.
d6 (plant_Machteshim, ramatHovav_ind) :: located_at_ramatHovav.
d7 (waste_NaCl, back_yard_Machteshim) :: buried_at.

Example 6 A DFL query answering session:

Queries:

```
-----  
-----  
?- q(( candidate_for_closing(P) , P :: plant )).  
?- q(( L :: risky_place )).
```

The answer to the first query is $P = \text{plant_Machtshim}$, and to the second query is $L = \text{back_yard_Machtshim}$. Both queries can be answered only by a collaborative effort of the two reasoners. The proof of the first query starts with rule $r8$, which is computed top-down, and produces the query $P :: \text{dangerous_plant}$. Since the predicate $::$ is tabled (common to both reasoners), it is computed bottom-up, and therefore triggers both reasoners. Neither reasoner can prove it independently. During the bottom-up computation of this query, most answers needed for other queries are tabled in the database. The answer for the second query is also tabled during the first proof. Following is part of the database after the first proof, by order of evaluation.

```
start runing  
  ansr located_at_ramatHovav < * located_at_BeerSeva  
  ansr located_at_BeerSeva < *  
    androle([located_at_south,located_at_desert])  
  ansr located_at_ramatHovav < *  
    androle([located_at_south,located_at_desert])  
  ans  plant_Machtshim ::  
    and([plant,all(produces,chemical_product)])  
  ans  waste_Br2 :: toxic_waste  
  ansr located_at_ramatHovav < * located_at_south  
  ansr located_at_BeerSeva < * located_at_south  
  ansr located_at_ramatHovav < * located_at_desert  
  ansr located_at_BeerSeva < * located_at_desert  
  ans  chemical_plant <= plant  
  ans  plant_Machtshim :: plant  
  ans  plant_Machtshim :: dangerous_plant  
  ans  (plant_Machtshim, ramatHovav_ind) :: located_at_BeerSeva  
  ans  (plant_Machtshim, ramatHovav_ind) ::  
    androle([located_at_south,located_at_desert])  
  ans  (plant_Machtshim, ramatHovav_ind) :: located_at_south  
  ans  (plant_Machtshim,ramatHovav_ind) :: located_at_desert  
  ans  (plant_Machtshim,ramatHovav_ind) ::  
    androle([located_at_desert])  
  ans  waste_NaCl :: toxic_waste  
  ans  dangerous_plant <= explodable
```

```

ans  plant_Machteshim :: explodable
ans  back_yard_Machteshim :: risky_place
ans  chemical_plant <=
      all(produces,chemical_product)
ans  plant_Machteshim :: all(produces,chemical_product)

```

unf-semi*** Goal is true ***

```

processing Time is: 0 minutes and 7 seconds
running Time is: 0 minutes and 1 seconds
total Time is: 0 minutes and 8 seconds
Total DB: 771
Number of Ans: 41
P = plant_Machteshim

```

```

| ?- q1(( L :: risky_place )).
start runing
running Time is: 0 minutes and 3 seconds
Total DB: 910
Number of Ans: 41
L = back_yard_Machteshim

```

Following are further results with a larger KB of 60 descriptions and 20 rules (of which the former is a part). Most time is devoted for preprocessing of the database.

```

| ?- q(( candidate_for_closing(P),
      R :: broken_plant )).

```

```

unf-semi*** Goal is true ***
processing Time is: 9 minutes and 18 seconds
running Time is: 1 minutes and 33 seconds
total Time is: 10 minutes and 51 seconds
Total DB: 5778
Number of Ans: 240

```

```

P = plant_Machteshim,
R = plant_Brom

```

```

| ?- q1(( R :: broken_plant,
      R :: all(pollution,risky))).

```

unf-semi*** Goal is true ***

running Time is: 0 minutes and 4 seconds
Total DB: 5801
Number of Ans: 240

R = plant_Brom

| ?- q1((R :: all(pollution,unrisky))).

unf-semi*** No ***
running Time is: 1 minutes and 40 seconds
Total DB: 6210
Number of Ans: 240

Cleaning the database from the answers that have been collected during the session results, for the same query:

unf-semi*** No ***
running Time is: 7 minutes and 37 seconds
Total DB: 5009
Number of Ans: 240