$\mathcal{DFL}$ – A Dialog Based Integration of Concept and Rule Reasoners

Mira Balaban and **Adi Eyal**
Ben-Gurion University of the Negev
ISRAEL

Abstract

$\mathcal{DFL}$ is a system that integrates a Description Logic 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 $\mathcal{DFL}$ 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.

1 Introduction

Description Logic (DL) is a collective name for knowledge representation formalisms that concentrate on the management of essential descriptive vocabulary ([2, 12, 15, 4]). The combination of a DL reasoner with reasoners in different paradigms is studied and implemented within several research efforts. The CLASSIC ([2]) and BACK ([10, 11]) systems include an "operational" rule component, and have features for reasoning under the *Closed World Assumption* (CWA). $\mathcal{AL}$-log ([5]) and CARIN ([9]) are frameworks for combining a DL reasoner with a DATALOG reasoner. The restricted rules of CLASSIC and BACK are used in an operational, system dependent manner. Interestingly, the epistemic logic $\mathcal{ALCK}$ defined in [6] provides an account for such rules. $\mathcal{AL}$-log and CARIN use a DL reasoner as an aid for a DATALOG reasoner, that cannot infer new descriptions on its own. In particular, in the $\mathcal{AL}$-log or CARIN frameworks there is no dialog between the components.

$\mathcal{DFL}$ is an implemented hybrid of a *description logic* reasoner and a *rule* reasoner. The rule language is *F(rames)-Logic* ([7, 8]), a rather expressive object-oriented language. Based on our previous work, where we proved that DLs are subsets of F-logic ([1]), F-Logic rules can include description formulae. Consequently, 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 latter has the flavor of theory revision for the DL component, since the rule component provides new description assertions (including intensional formulae), based on its own experience.

The $\mathcal{DFL}$ 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 (OWA)* for the DL reasoner, and the *Closed World Assumption (CWA)* for the Rule reasoner. $\mathcal{DFL}$ generalizes all former integrations of DLs with Rules, since it can support any number of Concept reasoners; the DL reasoners can reason about different description languages; the rule kinds include all rule types allowed in existing systems. Moreover, the general viewpoint of our framework leads to some free extensions of the rules of $\mathcal{AL}$-log and CARIN, i.e., extensions that do not complicate their functionality.

2 Architecture of $\mathcal{DFL}$

$\mathcal{DFL}$ is a knowledge base management system that supports integration of descriptions and object-oriented rules. In $\mathcal{DFL}$, the knowledge base manages a database of facts and of explicit descriptions in an arbitrary description language. The database consists of the following descriptions (object, concept and role terms are denoted o, c, and r, respectively):

1. Facts, in the form of predications $p(t_1, \ldots, t_n)$ (not necessarily ground).
2. Extensional ground assertions of the form $o \in c$, and $(o_1, o_2) \in r$.
3. Intensional ground descriptions of the form $c_1 \leq c_2$, $r_1 \leq r_2$, $c_n \doteq c$, and $r_n \doteq r$, where c_n and r_n are concept and role symbols, respectively.

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

1. *DL* – The *Description Languages* reasoner: A decidable reasoner, that reasons on the basis of the intended meaning of the description operators that form the descriptions.
2. *R* – The *Rule* reasoner, that reasons on the basis of given rules and some agreed upon semantical pol-

icy (e.g., perfect model for handling negation, or non-monotonic inheritance for handling default inheritance). The rules are F-Logic rules; their atoms are (not-necessarily ground) descriptions and predications.

This architecture is described in Figure 1.

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.

3 Semantics of $\mathcal{DFL}$

The semantics of $\mathcal{DFL}$, with respect to a given set P of DL constructors, is composed from the independent semantics of DL and R, which may operate along different reasoning policies. The semantics consists of a set of syntactic objects, either in DL terms, or in terms of the underlying F-Logic formalism. It assumes that the DL and the R reasoners are associated with semantic operators $\mathcal{DL}^P$ and $\mathcal{R}^{RULES}$, (written just $\mathcal{DL}$ and $\mathcal{R}$, for short), 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.

$\mathcal{DFL}$ is formally defined as follows:

Define: $T^{P,RULES}(D) \stackrel{def}{=} \mathcal{DL}^P(D) \cup \mathcal{R}^{RULES}(D)$
(Short notation: $T(KB)$)

and $T^0(KB) = KB^0$ – semantics dependent initial version of KB.

$$T^{k+1}(KB) = T(T^k(KB)) \quad k \geq 0$$

$$T^{\omega}(KB) = \bigcup_{k=0}^{\infty} T^k(KB)$$

Then: $\mathcal{DFL}(KB) \stackrel{def}{=} T^{\omega}(KB)$

The compositional semantics does not specify the $\mathcal{DL}^P$ and the $\mathcal{R}^{RULES}$ semantics. The declarative semantics is defined as the set of F-Logic models of $\mathcal{DFL}(KB) \cup RULES \cup FL_P$.

4 Inference in $\mathcal{DFL}$

In order to obtain a complete but focused inference, the interpreters that implement the DL and the R reasoners need to demonstrate the following behavior:

- Complete and goal directed interpreters.
- Interpreters document their failures.
- Interpreters have the capability to resume previously failed proofs. Preferably, memoization based interpreters.

The magic-set approach in logic programming simulates the control of a goal directed proof ([13, 14, 16]). Adding memoization yields the desired properties for the interpreters of the reasoners. A magic-set-with-memoization 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. Following this observation, the two reasoners in the $\mathcal{DFL}$ system are implemented by magic-set-with-memoization Prolog interpreters. The desired system behavior is obtained by using the global database for all memoizations. Each reasoner has its own theory, and the magic-set-with-memoization interpreter is applied, together, to the theory and the global database.

System operation is triggered by a conjunctive goal of descriptions and predications (not-necessarily ground). The system works out the goal as a Prolog goal, from left to right, relying on the unification mechanism of Prolog. Both interpreters try to prove the goal. The interpreters use the same global database for their memoizations. In order to keep the size of the database small, the interpreters perform controlled memoization. Only calls and answers to descriptive goals are memoized, since only descriptions 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. Operation is improved by using efficient interpreters.

Memoization based approaches usually trade time for space. In certain cases they replace infinite proof branches for finite proofs. In our approach to heterogeneous systems, which is implemented in the $\mathcal{DFL}$ system, memoization acts as the communication channel between the system reasoners. In particular, the memoization of failures is critical for the evolution of the dialog. We have tried to reduce the size of the database by generalizing the memoized goals. Clearly, we do not duplicate syntactic invariants (goals that are equal up to variable renaming). The memoization of goals cannot be further generalized to goals that do not subsume each other (in the sense of term subsumption in logic), since the DL reasoner needs ground goals, which are blocked by the subsumption test (once a non-ground goal like $call(a \leq X)$ is memoized, it blocks the memoization of more concrete goals). The database can be reduced by removing, during the proof, all "call" assertions that have matching answers. At the end of each proof session, all "call" assertions can be removed from the database.

4.1 The DL Reasoner

The structural subsumption approach is less powerful than the constraint approach, since there are no complete subsumption and instance checking algorithms for powerful languages (CLASSIC's algorithm is complete, though [3]). Moreover, structural subsumption restricts

$\mathcal{DFL}$ MANAGER

$\nearrow\swarrow$ $\searrow\nwarrow$

R : *Consults RULES*	$\leftrightarrow$	D – A DATABASE of FACTS and DESCRIPTIONS	$\leftrightarrow$	DL : *Consults DESCRIPTION OPERATORS*

Figure 1: Architecture of a $\mathcal{DFL}$ Knowledge Base

the terminology to be acyclic, while constraint methods apply to cyclic terminologies. However, structural subsumption is reductional in nature, and hence, fits well into the heterogeneous framework of $\mathcal{DFL}$. In addition, it is easier to implement structural subsumption than tableau methods. The DL interpreter in the $\mathcal{DFL}$ system is a variant of the structural subsumption approach. The deviations from the conventional approach emerge from the requirements set by the dialog between the reasoners. Structural subsumption based systems usually operate under the *expand-normalize-compare* method. The $\mathcal{DFL}$ algorithm saves some provably unnecessary expansions.

Claim 1 *If the terminology includes no definitional cycles, and no cycles are generated by the R reasoner, then under a memoization policy the $\mathcal{DFL}$ algorithm is guaranteed to terminate.*

4.2 The R Reasoner

The rules reasoner is implemented by a magic-set-with-memoization interpreter that is extended with "selective" unfolding on the rules. The purpose of the unfolding is to let R infer also 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 "selectivity" of the unfolding is intended to prevent rule explosion. The R reasoner performs unfolding only on rules whose head can be the head of a role inclusion formula, i.e., $(X,Y) \in role\_expression$. This way, the focused nature of the magic-set interpreter is not lost.

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.

4.3 Results

The following example demonstrates a query whose answer requires an intensive dialog between the two reasoners. Note that the rules include intensional descriptions, and that descriptions appear both in the body and in the heads of rules. They extend the kinds of rules supported by BACK, CLASSIC, CARIN, and $\mathcal{AL}$-log. The example also demonstrate the combination of different reasoning policies: Closed world for rules, and open world for descriptions. Notation: :: stands for membership, <= stands for concept inclusion, $<*$ stands for role inclusion, and <=> stands for concept equality.

Example 1 A $\mathcal{DFL}$ query answering session*:*

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

r2) Y :: dangerous_plant :-
        Y :: plant,
        (Y,X)::LOCATED_AT, %LOCATED_AT is a variable.
        X :: near_population,    % A risky_place.
        LOCATED_AT <* located_at_desert.

r3) X::toxic_waste:-X::and([waste,chemical_material]).

r4) X :: toxic_waste :- (P,X) :: produces,
           P :: dangerous_plant, (X,Z) :: buried_at.

r5) (X,L) :: located_at_BeerSheva :-
                          (X,L) :: located_at_ra.

r6) (X,L)::androle(
           [located_at_south,located_at_desert]):-
                     (X,L) :: located_at_BeerSheva.

r7) Plant_Type<=explodable :-
                temperature(Plant_Type,60).

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).
    all_known and the negation are CWA predicates.

r9) temperature(dangerous_plant,60).

r10) no_gov_control( plant_Ma).

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

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

--------------------------------------------
Data Base:
--------------------------------------------
d1  (plant_Ma,waste_NaCl) :: produces.
d2  plant_Ma :: chemical_plant.
d3  chemical_plant <=>
   and([plant, all(produces,chemical_product)]).
d4  waste_Br2 :: and([waste,chemical_material]).
d5  ra_ind :: near_population.
d6  (plant_Ma, ra_ind) :: located_at_ra.
d7  (waste_NaCl, back_yard_Ma) :: buried_at.
--------------------------------------------
Queries:
--------------------------------------------
?- q(( candidate_for_closing(P) , P :: plant  )).
?- q(( L :: risky_place )).
```

The answer to the first query is P= plant_Ma, and to the second query is L = back_yard_Ma. 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 :: *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 this query, most answers needed for other queries are memoized in the database. The answer for the second query is also memoized during the first proof. Following is part of the database after the first proof, by order of evaluation.

```
start runing
  ansr located_at_ra <* located_at_BeerSeva
  ansr located_at_BeerSeva <*
   androle([located_at_south,located_at_desert])
  ansr located_at_ra <*
   androle([located_at_south,located_at_desert])
  ans  plant_Ma ::
    and([plant,all(produces,chemical_product)])
  ans  waste_Br2 :: toxic_waste
  ansr located_at_ra <* located_at_south
  ansr located_at_BeerSeva <* located_at_south
  ansr located_at_ra <* located_at_desert
  ansr located_at_BeerSeva <* located_at_desert
  ans  chemical_plant <= plant
  ans  plant_Ma :: plant
  ans  plant_Ma :: dangerous_plant
  ans  (plant_Ma,ra_ind) :: located_at_BeerSeva
  ans  (plant_Ma,ra_ind) ::
    androle([located_at_south,located_at_desert])
  ans  (plant_Ma,ra_ind) :: located_at_south
  ans  (plant_Ma,ra_ind) :: located_at_desert
  ans  (plant_Ma,ra_ind) ::
                    androle([located_at_desert])
  ans  waste_NaCl :: toxic_waste
  ans  dangerous_plant <= explodable
  ans  plant_Ma :: explodable
  ans  back_yard_Ma :: risky_place
  ans  chemical_plant <=
                  all(produces,chemical_product)
  ans  plant_Ma :: 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_Ma

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

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_Ma,
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 qury:

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

5 Discussion

The BACK ([10, 11]) and CLASSIC ([2]) systems support rules of the form: $X \in c_0 \longleftarrow X \in c_1$. The direction of interaction is from the rules to the descriptions. In $\mathcal{AL}$-log ([5]) and CARIN ([9]) descriptions are added to DATALOG (function free) rules in a Constraint Logic Programming style. Rule heads cannot be description formulae, but the atoms in a rule body can be description formulae, of a restricted type[1]. Hence, the direction of interaction is from descriptions to rules. In particular, CLASSIC and BACK like rules, are not possible in $\mathcal{AL}$-log and CARIN. Case analysis over descriptions in rule bodies is possible (Not possible in $\mathcal{DFL}$).

In the $\mathcal{DFL}$ system, efforts to **extend** the expressivity of description logics are replaced by efforts to **combine** them with other reasoning paradigms. To the best of our knowledge, the $\mathcal{DFL}$ 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 leads to natural extensions of previous integrations of rules and descriptions (e.g., CARIN and $\mathcal{AL}$-log). The dialog situation leads to the development of a structural subsumption based DL interpreter, that achieves meaningful saving in symbol expansions, though at a small lose of expressive power. The saving obtained by our algorithm, when applied to real problems, still needs to be empirically evaluated.

The $\mathcal{DFL}$ system can be extended to integrate a variety of reasoners, including multiple DL reasoners, CARIN or $\mathcal{AL}$-log like reasoners, several rule reasoners, and reasoners of other types. Achieving a general integration is demanding, since it requires an exact characterization of the syntactic overlaps between the information sources, and assumes the availability of interpreters that have the properties characterized in this paper. Nevertheless, once such interpreters are given, and their interaction analyzed, the $\mathcal{DFL}$ architecture can support their integration into a broad heterogeneous system.

References

[1] M. Balaban. The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, 15:19–60, 1995.

[2] A. Borgida, R. Brachman, D. McGuinness, and L. Resnick. Classic: A structural data model for objects. In *ACM-SIGMOD-89*, Portland, OR, 1989.

[3] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, 1:277–308, June 1994.

[4] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class-based representation formalisms. In *KR-94*, pages 109–120, 1994.

[5] 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.

[6] F. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. Adding epistemic operators to concept languages. In *KR-92*, pages 342–353, 1992.

[7] M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *SIGMOD-89*, 1989.

[8] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.

[9] A. Levy and M.-C. Rousset. Combining horn rules and description logics in carin. *AIJ*, 104(1-2):165–209, 1998.

[10] 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.

[11] 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.

[12] 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.

[13] H. Tamaki and T. Sato. Old resolution with tabulation. In *3rd International Conference on Logic Programming*, pages 84–98, 1986.

[14] J. Ullman. *Principles of Database and Knowledge-base Systems*. Computer Science Press, 1989.

[15] 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.

[16] J. Wunderwald. Memoing evaluation by source-to-source transformation. In *LOPSTR-95*, 1995.

[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.