Abstract

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 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 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. 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 AC-log and CARIN, i.e., extensions that do not complicate their functionality.

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). AC-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 ALCX defined in [8] provides an account for such rules. AC-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 AC-log or CARIN frameworks there is no dialog between the components.

DFL is an implemented hybrid of a description logic reasoner and a rule reasoner. The rule language is Frames-Logic ([7, 8]), a rather expressive object-oriented language. Based on our previous work, where we proved that DLs are subsets of F-logic ([11]), 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 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. 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 AC-log and CARIN, i.e., extensions that do not complicate their functionality.

2 Architecture of DFL

DFL is a knowledge base management system that supports integration of descriptions and object-oriented rules. In 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(t₁, . . . , tₙ) (not necessarily ground).
2. Extensional ground assertions of the form o ∈ c, and (o₁, o₂) ∈ r.
3. Intensional ground descriptions of the form c₁ ≤ o₂, r₁ ≤ r₂, cₙ = c, and rₙ = r, where cₙ and rₙ 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-
This architecture is described in Figure 1.

While in query mode the 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 DFL

The semantics of 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 R reasoners are associated with semantic operators $D^P$ and $R^{RULES}$ (written just $D$ and $R$, 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 $D^P$ and $R^{RULES}$ to $D$. Thus, the principles of modularity and compositionality are kept.

DFL is formally defined as follows:

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

and

$$ T^c(KB) = KB^c $$  - semantics dependent
$$ k \geq 0 $$

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

Then:

$$ DFL(KB) \stackrel{def}{=} T^c(KB) $$

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

4 Inference in 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.

Adding memoization yields the desired properties of 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 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 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 substitution in logic), since the DL reasoner needs ground goals, which are blocked by the subsumption test (once a non-ground goal like call(a ≤ 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
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 DFL. In addition, 
it is easier to implement structural subsumption than 
tableau methods. The DL interpreter in the 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 rea-
soners. Structural subsumption based systems usually 
operate under the expand-normalize-compare method. 
The DFL algorithm saves some provably unnecessary 
expansions.

Claim 1 If the terminology includes no definitional cy-
cles, and no cycles are generated by the R reasoner, then 
under a memoization policy the DFL algorithm is guar-
anteed to terminate.

4.2 The R Reasoner
The rules reasoner is implemented by a magic-set-with-
memorization 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 “selectiv-
ity” of the unfolding is intended to prevent rule explo-
sion. 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 in-
heritance, negation, or types.

4.3 Results
The following example demonstrates a query whose an-
swer requires an intensive dialog between the two reason-
ers. 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 AC-log. The exam-
ple also demonstrate the combination of different reason-
ring policies: Closed world for rules, and open world for 
descriptions. Notation: \( \leq \) stands for concept inclusion, \( \ast \) stands for role inclu-
sion, and \( \iff \) stands for concept equality.

Example 1 A 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 \ast 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<=explosable :-
   temperature(Plant_Type,60).
r8) candidate_for_closing(P):-
   P :: dangerous_plant,
   dangerous_plant \leq explosable,
   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).
Data Base: (plant_Ma, waste_NaCl) : produces.
plant_Ma : chemical
plant

chemical_plant <>
and({plant, all(produces, chemical_product)}).
waste_br2 : and([waste, chemical_material]).
ra_ind : near_population.
plant_Ma, ra_ind : located_at.ra.
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 running
ans located_at_ra <- located_at_BeerSeva
ans located_at_BeerSeva <-
    androle([located_at_south, located_at_desert])
ans located_at_ra <-
    androle([located_at_south, located_at_desert])
ans plant_Ma ::
    and([plant, all(produces, chemical_product)])
ans waste_br2 :: toxic_waste
ans located_at_ra <- located_at_south
ans located_at_BeerSeva <- located_at_south
ans located_at_BeerSeva <- located_at_south
ans located_at_BeerSeva <- located_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 waste_NaCl :: toxic_waste
ans back_yard_Ma :: risky_place
ans chemical_plant <-
    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
L = back_yard_Ma

?- q(( L :: risky_place )).
start running
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

?- q(( 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

?- q(( R :: all(pollution, unrisky))).

unf-semi*** No ***

running Time is: 1 minutes and 40 seconds
Total DB: 6210
Number of Ans: 240
5 Discussion

The BACK ([10, 11]) and CLASSIC ([2]) systems support rules of the form: \( X \in c_0 \iff X \in c_1 \). The direction of interaction is from the rules to the descriptions. In AC-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. Hence, the direction of interaction is from descriptions to rules. In particular, CLASSIC and BACK like rules, are not possible in AC-log and CARIN. Case analysis over descriptions in rule bodies is possible (Not possible in DFL).

In the 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 DFL system is the first to support a true dialog between a DL and a rule reasoner, 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 AC-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 loss of expressive power. The saving obtained by our algorithm, when applied to real problems, still needs to be empirically evaluated.

The DFL system can be extended to integrate a variety of reasoners, including multiple DL reasoners, CARIN or AC-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 DFL architecture can support their integration into a broad heterogeneous system.

References


