Augmenting Knowledge Representation and Reasoning Languages with Customizable Metalogic Features

Stefania Costantini Andrea Formisano

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila

DIPARTIMENTO DI MATEMATICA E INFORMATICA

CILC 2019, Trieste,
June 20, 2019

Ontological Reasoning in OWL

Relevant aspects concerning knowledge representation and reasoning that can be found in OWL are:

- properties of relations: e.g., symmetry/asymmetry, transitivity, functionality, reflexivity/irreflexivity, domain/range;
- relations between relations (meta-properties): e.g., inverse-of, equivalence, disjointness, subclass;
- cardinality of relations.

A *Reasoner* draws logical consequence from an ontology given its contents and the definition of such properties

Our Proposal

Empower logic-based knowledge engineering, representation and reasoning languages with OWL-like features

- we employ meta-level axiom schemata based upon a naming (reification) device
- we propose a method for extending the semantics accordingly.
- we improve over OWL as we allow for user-defined new properties

As a proof of concept we consider Answer Set Programming (ASP)

Metarules and Reflection

$solve(P'(X', Y')) :- symmetric(P'), solve(P'(Y', X'))$.
$symmetric(friend')$.
$friend(anne, carl)$.

$solve\_not(P'(X', X')) :- irreflexive(P')$.
$irreflexive(friend')$.

Metarules and Reflection: How it works

- At the object level, $friend(carl, anne)$ or $friend(carl, carl)$ attempted as usual, and may fail
- At the metalevel, both $solve(friend'(carl', anne'))$ and $solve\_not(friend'(carl', anne'))$ are attempted.
- Success of *solve* enables action execution, success of *solve_not* prevents it (where *solve_not* has priority over *solve*).

Other Examples

$solve(P'(X', Y')) :- transitive(P'), solve(P'(X', Z')), solve(P'(Z',$

$transitive(same\_age')$.
$same\_age(ann, alice)$.
$same\_age(alice, chris)$.
%derived: $same\_age(ann, chris)$

$solve(P'(X', Y')) :- equivalent(P', R'), solve(R'(X', Y'))$.

$equivalent(friend', amico')$.
$symmetric(equivalent')$.
$friend(ann, alice)$.
%derived: $amico(ann, alice)$

The meta-meta statement $symmetric(equivalent')$ allows the translation to be applied both ways.

Metarules and Reflection for Run-time agents'self-checking

$solve(execute\_action'(Act')) :-$
$\quad present\_context(C), ethical(C, Act').$

$solve\_not(execute\_action'(Act')) :-$
$\quad present\_context(C), ethical\_exception(C, Act').$

% *We might have e.g., one of the following* :
$context(reality).$
$context(videogame).$
$context(storytelling).$
. . . or others

Background: Answer Set Programming (ASP)

- Answer Set Programming (ASP), is a well-known successful logic programming paradigm
- Roughly speaking, an ASP program is a declarative Prolog-like (executable) specification of a problem to be solved.
- Such a program may have several "models", called "answer sets" (or also "stable models"), each one representing (if any exists) a possible interpretation of the situation described by the program (and, usually, encoding a solution to the problem at hand).

Background: Answer Set Programming, cont'd

The ASP programming methodology can be called GCO, for "Guess & Check & Optimize", where:

- Guess implies generating potential solutions via rules and cycles;
- Check implies selecting admissible ones by defining suitable constraints;
- Optimize implies specifying preference criteria by exploiting *weak constraints*, indicated by connective $:\sim$, that select among the admissible solutions those that satisfy such constraints at best.

Several "ASP solvers" are freely available.

Done in ASP so far: HEX Programs

Higher-order syntax with polynomial reduction to standard syntax $C(X) :- subClassOf(D, C), D(X).$

So-called "External computations" that influence (a) the instantiation of the program (by performing value invention), and (b) the solving process

Attempts to provide lightweight and efficient evaluation machinery (e.g., Hexlite solver at JELIA 2019 by Peter Schüller)

Sample atoms to be evaluated in an external ontology:

- $\&transitive[p]()$ of type (S), to verify whether $p/2$ is transitive
- $\&rdf[U](S, P, O)$ to import rdf triples from URL U (meta-properties applied in the external ontology)

Our attempt

Perform elaboration considering properties of relations "in-house"

Some properties (e.g., those expressible in OWL) possibly predefined, other properties user-defined

Our attempt: how-to

We compile a set S of metalevel and metaevaluation rules into a form that can be seamlessly added to a given ASP program Π, whose rules also undergo some easy modifications; distinction between predicate and constant symbols is maintained.

We thus obtain an augmented program Π^S where necessary conditions are satisfied to ensure that properties of relations specified in S are properly applied.

Linear transformation from Π to Π^S (size of Π^S at worst three times the size of Π)

Our attempt: how-to (cont'd)

Basically, we rewrite Π into Π^S such that the answer sets of Π^S satisfy the axiom schemata:

$$A \leftarrow solve(A')$$
$$\neg A \leftarrow solve\_not(A')$$

where the second one has priority and we have that

$$\frac{A}{solve(A')}$$

Our attempt: how-to (cont'd)

We have been able to prove the following:

Theorem
Properties in S (and their combinations) are satisfied in the answer sets of Π^S.

"Result sets" for given program Π are extracted from the answer sets of Π^S by deleting metaevaluation atoms (atoms concerning *solve* or *solve_not*).


Comparison with HEX Programs

No semantic distinction between predicate and constant symbols

$$P(X,Y) :- symmetric(P), P(Y,X).$$
$$friend(ann, mary).$$
$$symmetric(friend).$$

Four symbols implies 4! ground rules (w.r.t. just 4 in our case) among which weird ones, such as:

$$mary(friend, anne) :- symmetric(mary), mary(anne, frien$$

No problem with given fact, "correct" answer sets (contents as expected, $friend(mary, ann)$ derived)

Comparison with HEX Programs (cont'd)

No definition is possible for irreflexive, asymmetric etc. (no counterpart of *solve_not*) and, on examples such as:

$P(X, X) :- reflexive(P).$
$friend(ann, mary).$
$reflexive(P) :- not\ irreflexive(P).$
$irreflexive(P) :- not\ reflexive(P).$

there is a combinatorial explosion of weird answer sets, containing for instance

$reflexive(anne), anne(mary, mary), \ldots$ or
$irrreflexive(anne), reflexive(mary), mary(friend, friend), \ldots$


Future Directions

- Extension of the approach to Datalog^{+-}
- In fact, Datalog^{+-} has been shown suitable for ontological reasoning, but limited to the object-level
- Needed: modify the Chase procedure for drawing consequences from metalevel rules also

Thank you for your attention!
Questions are welcome