Combining OWL with F-Logic Rules and Defaults

Heiko Kattenstroth, Wolfgang May, Franz Schenk

Institut für Informatik, Universität Göttingen, Germany

{hkattens,may,schenk}@informatik.uni-goettingen.de

Supported by the EU Network of Excellence

ALPSWS 2007, Porto, Portugal, Sept. 13, 2007

SWAN: Semantic Web Application Node

- an application node in the Semantic Web (airline service, university, . . .)
  deployed in an ECA-(Event-Condition-Action-)Rules environment: raises events, executes actions
- knowledge base (facts + x)
- potentially large fact base
- ontology given in OWL $\Rightarrow$ some built-in reasoning
- intensional knowledge by rules
  - derived relationships
  - closed world negation
- nonmonotonic/default inheritance

DL and Rules

Attractive Combination

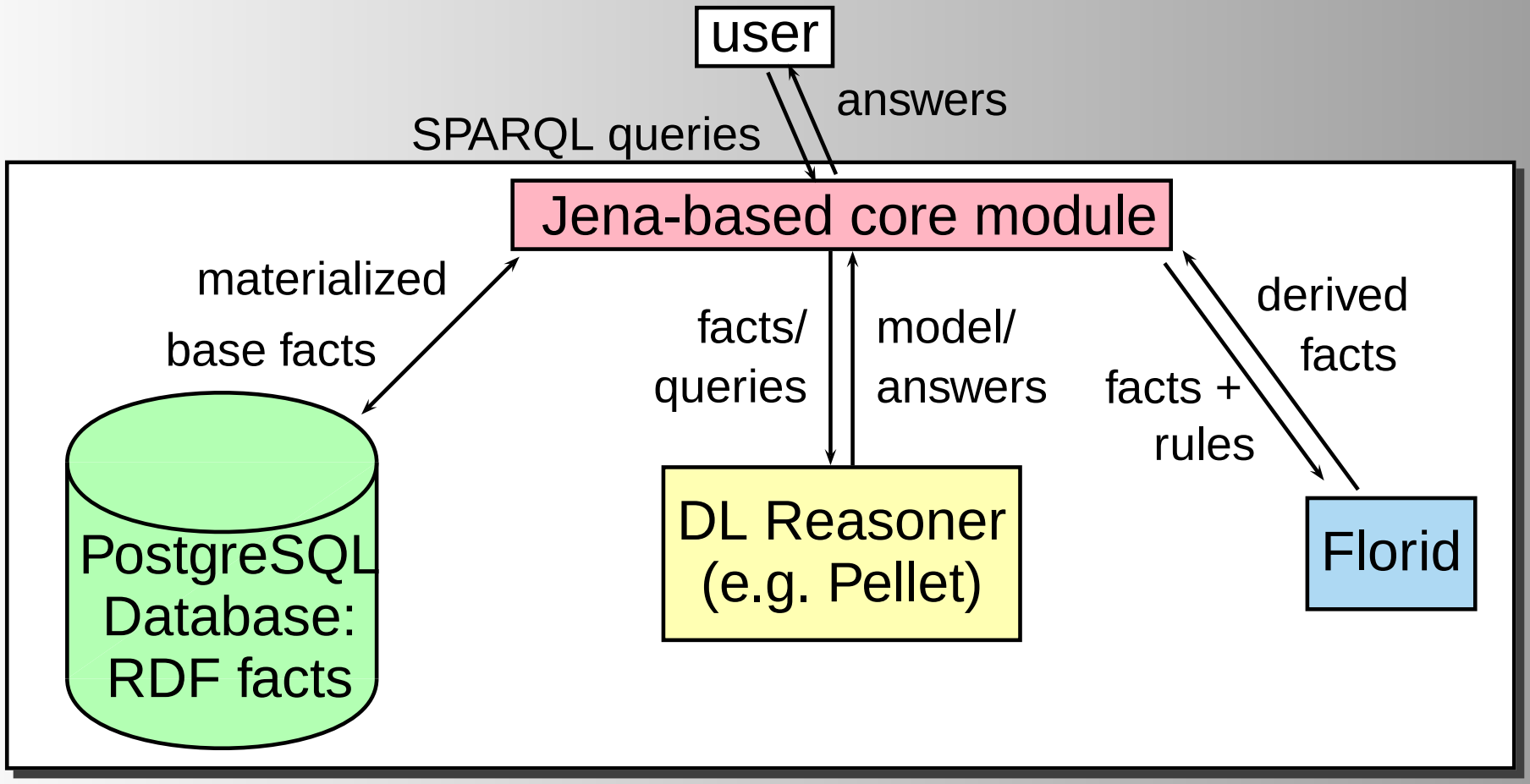
- LP: efficient; bottom-up, top-down, stratified, well-founded; polynomial; ASP; closed-world
- DL: open-world; implicit existential knowledge; has a very restricted, but nevertheless useful decidable fragment (tree property of expressions; two-variables) $\mathcal{SHOIQ}(\mathcal{D})$, OWL-DL

Many approaches for DL+rules have been presented so far

- CARIN (1995), DL-log (1998), many recent approaches
- Decidability: DL-safeness of rules (implicitly known objects)
- high complexity

Motivation and Architecture of DL+Florid

- pragmatical solution based on the tools we have
- highly compatible data model: $(x\ r\ y) \sim x[r \rightarrow y]$
- combine Jena+OWLReasoner with Florid as a “slave”



Tight coupling

- DL part and the rule part are not required to be disjoint
- rules can be used to derive concept memberships and role instances and to create objects (URIs)
- each concept and relationship can be used in OWA context and in CWA context
- (DL + (Rules)*)* alternating fixpoint computation
- optionally topped with F-Logic's default inheritance

F-Logic $\leftrightarrow$ DL/RDF/OWL

	F-Logic	DL	RDF/OWL
is-a atoms	$o : c$	$C(o)$	o a c
	$c :: d$	$C \sqsubseteq D$	c rdfs:subClassOf d
signature/	$c[p \Rightarrow d]$	$C \sqsubseteq \forall p.D$	rdfs:range
range	$c[p \Rightarrow\!\!\!> d]$	”	”
scalar prop.	$o[p \rightarrow v]$	$p(o,v)$	$o\ p\ v$
multivalued prop.	$o[p \rightarrow\!\!\!> \{v_1,\ldots,v_n\}]$	$p(o,v_1),\ldots,p(o,v_n)$	$o\ p\ v_1,\ldots,v_n$
default	$c[p \bullet\!\!\rightarrow v]$	–	–
inheritance	$c[p \bullet\!\!\rightarrow\!\!\!> \{v_1,\ldots,v_n\}]$	–	–
identifiers	names, URIs, literals	names, literals	URIs, blanks, literals

Pure facts can easily be transformed:

“foo:bla#john”:“foo:meta#Person”[“foo:meta#name”$\rightarrow$“John”;
“foo:meta#livesIn”$\rightarrow$(“geo://de/Berlin”:“geo:meta#City”)].

F-Logic Semantics: bottom-up T_P^ω + add-ons

- Herbrand-style “F-structures”
- built-in axioms $\mathcal{C}\ell(\mathcal{H})$: isa and subclass transitivity, uniqueness of scalar properties ($\rightarrow$ equating objects)

$$T_P(\mathcal{H}) := \mathcal{H} \cup \{h \mid (h \leftarrow b_1, \ldots, b_n) \text{ is a ground instance of a rule of } P \text{ and } b_i \in \mathcal{H} \text{ for all } i = 1, \ldots, n\},$$

$$T_P^0(\mathcal{H}) := \mathcal{C}\ell(\mathcal{H}),$$

$$T_P^{i+1}(\mathcal{H}) := \mathcal{C}\ell(T_P(T_P^i(\mathcal{H}))),$$

$$T_P^\omega(\mathcal{H}) := \begin{cases} \lim_{i\to\infty} T_P^i(\mathcal{H}) & \text{if } T_P^0(\mathcal{H}), T_P^1(\mathcal{H}), \ldots \text{ converges,} \\ \bot & \text{otherwise.} \end{cases}$$

- User-stratified: Perfect Model $T_{P_n}^\omega(\ldots(T_{P_1}^\omega(\emptyset)))$.

Things to be expressed by Rules

- Positive rules: derived properties, properties with attributes (e.g., “connection from x to y with duration t”)
- CWA negation
- aggregation
- stratification: operational flavor e.g. for data integration
- query answering: more efficient (polynomial) than DL reasoning
- e.g. compute shortest connections by stratification and aggregation (min).

Example: Query Answering

Compute all countries that have at least two cities with more than 1.000.000 inhabitants.

```
:isProvinceOf rdfs:subpropertyOf :belongsTo.
:cityIn rdfs:subpropertyOf :belongsTo.
:belongsTo a owl:TransitiveProperty;
      owl:inverseOf :hasProvOrCity. ## bridge country-prov-city
:Million a owl:DataRange;
      owl11:onDataRange xsd:int; owl11:minInclusive 1000000.
:HasBigPopulation owl:equivalentClass [a owl:Restriction;
      owl:onProperty :population; owl:someValuesFrom :Million].
:BigCity owl:intersectionOf (:City :HasBigPopulation).
:CountryWithTwoBigCities owl:intersectionOf (mon:Country
      [a owl:Restriction; owl:onProperty :hasProvOrCity;
      owl:minCardinality 2; owl11:onClass :BigCity]).
```

The actual evaluation by a reasoner takes some minutes.

Example: Query Answering

Cty:“geo:meta#BigCity” :-
    Cty:“geo:meta#City”, Cty[geo:meta#population”$\rightarrow\!\!\!\!\rightarrow$Pop], Pop $>$ 1000000.

C[“geo:meta#hasBigCity”$\rightarrow\!\!\!\!\rightarrow$Cty] :-
    C[“geo:meta#hasCity”$\rightarrow\!\!\!\!\rightarrow$Cty], Cty:“geo:meta#BigCity”.

C[“geo:meta#hasBigCity”$\rightarrow\!\!\!\!\rightarrow$Cty] :-
    C[“geo:meta#hasProvince”$\rightarrow\!\!\!\!\rightarrow$Prov], Prov[“geo:meta#hasCity”$\rightarrow\!\!\!\!\rightarrow$Cty],
    Cty:“geo:meta#BigCity”.

X:“geo:meta#CountryW2BigCities” :-
    X:“geo:meta#Country”, X[“geo:meta#hasBigCity” $\rightarrow\!\!\!\!\rightarrow\{$C1,C2$\}$], not C1 = C2.

- rule-based answer view defined on RDF/OWL fact base

Working with DL+Florid

Input: (L, P, D):

1. OWL ontology L: OWL concept (TBox) reasoning + instance (ABox) reasoning [theory reasoning],
2. a set P of rules: mainly about instances, but also concept derivation [T_P, Herbrand-style ground facts],
3. a set D of inheritance atoms (defaults)

Semantics

- OWL: a theory
- OWL+Rules: an undecidable theory with in general infinite universe (cf. Person $\sqsubseteq \exists 2$hasAncestor.Person) and infinitely many atomic facts about these
- Defaults: extensions of a default theory
- F-Logic: a (finite) set of ground facts
- SPARQL: queries

Target Semantics

$\mathcal{G}\ \mathcal{F}$:= atomic $\mathcal{G}$ round $\mathcal{F}$ acts of some Default Extension of $Th(L_{FOL} \cup P_{Comp})$ for the named objects from L's ABox and P's active domain and the constants introduced in D.

Alternating Fixpoint OWL+Rules

1. Materialize the OWL model of L (ABox + TBox).
   Due to its open-world nature, OWL/DL reasoning has a very sceptical negation. Doing OWL reasoning first is thus completely safe.
2. Application of deductive rules.
   export all (relevant, explicitly known) facts together with the rules P to Florid, apply bottom-up-evaluation.
   The (relevant) resulting facts are sent back to the Jena-based core.
   In case of only positive rules, it is again completely safe.

Iteration: the above steps are iterated until no new facts can be derived.

Implicitly known objects cause some trouble (als always for DL+Rules)

Data Exchange Jena → Florid

- Materialize OWL model
- export triples: facts as far as covered by the F-Logic data model: isa, subclass, (ranges), object properties,
- URI as subclass of string for object identifiers
- only explicitly known objects can be used directly:
  Parent $\sqsubseteq \exists$hasChild.Person vs. alice[hasChild →bob].

Pitfalls

- positive rules: potentially “incomplete” results
- negation: implicit things “don’t exist”

Data Exchange Jena $\rightarrow$ Florid (Details)

- x rdf:type c where c is an application class (as x_t:c_t),
- c rdfs:subClassOf d (as c_t::d_t)
- p rdfs:range c where p is a non-RDF/RDFS/OWL property (as $\langle$owl:Thing$\rangle_t[p_t \Rightarrow c_t]$ (if p is known to be functional) or $\langle$owl:Thing$\rangle_t[p_t \Rightarrow\!\!\!\!\!\rightarrow c_t]$ (otherwise)),
- $x_t\ p_t\ y_t$ where p is a non-RDF/RDFS/OWL property (as $x_t[p_t \rightarrow y_t]$ (if p is known to be functional) or $x_t[p_t \rightarrow\!\!\!\!\!\rightarrow y_t]$ (otherwise)).
- for literals ℓ, ℓ_t is the string representation of ℓ
- URIs are exported as elements of the class url::string, e.g., “http://www.w3.org/2002/07/owl#Thing”:url

```
“foo:meta#Person”:url.    “foo:meta#name”:url.    “foo:meta#livesIn”:url.
“foo:meta#john”:url.      “geo://de/Berlin”:url.
(“foo:bla#john”:“foo:meta#Person”)[“foo:meta#name”→“John”;
        “foo:meta#livesIn”→“geo://de/Berlin”].
```

Data Exchange Florid → Jena

inverse direction:

- export only atoms/triples x:c, c::d, $x[p \rightarrow y]$, $x[p \twoheadrightarrow y]$ where x, c, d, p are members of the class url; y may be a url or a literal.
- allows to have auxiliary names (non-URLs) local to Florid
- allows to introduce new concepts (as URLs) by Florid

Back in Jena:

- merge with original data
- if new facts have been derived, iterate alternating process until a fixpoint is reached.

Alternating Fixpoint (OWL+Rules)+Defaults

1. Compute fixpoint of DL+Rules. Store model $\mathcal{M}_0$
2. Apply one default rule for all its instances
3. Apply fixpoint of DL+Rules
   if consistent, store as $\mathcal{M}_{i+1}$ and continue with (2), otherwise reset to $\mathcal{M}_0$,
   4. Apply the default rule only to one instance
   5. Apply fixpoint of DL+Rules
      if consistent store as $\mathcal{M}_{i+1}$, continue with (4)
      otherwise block this instance, reset to $\mathcal{M}_i$, continue with (4)

Iteration: continue with (2).

Theoretical Properties

Positive Rules

- if P only consists of positive rules, everything is safe. The result is a subset of $\mathcal{G}\,\mathcal{F}$.
- objects that are only implicitly known are not used in rules.
- (other hybrid approaches also restrict this)
- current workaround: export implicit “border objects” as dummies (which are no URLs and thus not reimported),

$\Rightarrow$ ongoing work.

- If the implicit objects are handled appropriately, the result is correct and complete wrt. (some) $\mathcal{G}\,\mathcal{F}$.

Theoretical Properties

Rules with Negation

- note: OWA of OWL does not conflict with any facts that become known later,
- conflict: CWA of LP negation conflicts with the possibility of later derivation of positive facts by OWL reasoning.
- allows e.g. to use the negation of all base facts,
- can be traced by monitoring Σ^{-}.

Additional Florid Features

Use Florid not only for reasoning, but also for data extraction and procedural help:

- parse HTML or XML data by (X:url).get,
  wrap data into RDF by rules,
  add it to the facts $\rightarrow$ Jena.
- match regular expressions:
  match(*in-var*, *regexp*, n *groups*, n *out-vars*)
- execute perl predicate: perl(*command*,*in-vars*,*out-vars*)

Miscellanous Issues

- OWL-DL part empty (or a simple schema+data style database): semantics = F-Logic with default inheritance (well-behaved; cf. [May-Kandzia-JLC-01]).
- F-Logic rule part empty: OWL-DL + default inheritance. Again, inheritance is controlled in such a way that it inherits only when the application is consistent.
- class hierarchy: after inheriting, the default applications must not be invalidated
  (the program can be augmented dynamically to derive inconsistency in such cases)

Miscellaneous Issues

Loose Coupling between LP and DL

If information flow between both kinds of languages in only in one direction, the semantics is correct and complete for stratified negation:

- if the predicates occur in rule heads are disjoint from those of the OWL ontology (rule part only queries the ontology (as e.g. in DLVhex).
- if the predicates that occur in rule *bodies* are disjoint from those of the OWL ontology (rule part serves for populating the ontology, e.g., by information extraction from Web pages).

Conclusion

- OWL-DL + general F-Logic Rules + inheritance:
  - declarative alternative to programming some application-specific solution,
  - requires careful design of rules to be “provable correct”,
  - allows for rapid prototyping and flexible adaptation of the rules.
- Performance: not a runtime-reasoner, but intended to (re)compute the local state of an application after an update.
- Open issue: enhance the handling of existential anonymous objects.
  Currently, “dummies” in “one step” distance to named objects are generated for Florid as a workaround.

Conclusion

- OWL-DL + general F-Logic Rules + inheritance:
  - declarative alternative to programming some application-specific solution,
  - requires careful design of rules to be “provable correct”,
  - allows for rapid prototyping and flexible adaptation of the rules.

Demo available at http://www.semwebtech.org/DLFlorid

Questions?