Chapter 11
OWL Profiles, Rule-Based Reasoning, and Handling Reasoning with the Jena API

Types of Reasoning

- DL-Reasoner: Tableau Reasoning, FOL-based
- Rule-Based Reasoning (cf. "Deductive Databases" Lecture)
  - RDFS is purely rule-based
  - Inverse, Symmetry, and Transitivity are rule-based
  - Functionality (maxCard = 1) application for equating objects is rule-based
  - the above are all *purely positive rules*
  - Negation in the body: CWA (incl. stratification and WFS) vs. OWA
  - Disjunction in the head/choices
    (is also needed for cardinalities $>$ 1 and some other things)
    $\Rightarrow$ Stable Models

DL/Tableaux vs. Deductive Databases/Datalog

- DL-Reasoner: Tableau Reasoning
  - can be extended easily with additional Tableau Rules
  - main problem: strategies, blocking, ...
  - strategies detect where exponential growth of the tableau occurs. Try to keep polynomial if possible.
- Rule-Based Reasoning (cf. “Deductive Databases” Lecture)
  - Prolog/Datalog (Resolution Calculus, “backward-chaining”)
  - Bottom-up database completion (“forward-chaining”, T_P^ω-operator, optimizations like “seminaive evaluation”, “magic sets”)
  - not restricted to special constructs (like DL/Tableaux), can handle rules in general (cf. SWRL)
  - Specific problems with negation (Closed-World), stratified/well-founded semantics
  - disjunction in the head: no classical rules, only via stable models (which is basically more related to Model Checking and Tableaux)
  - well-suited for ABox reasoning (data, databases), less well-suited for TBox reasoning (*theorem proving*, ontologies)

OWL Profiles

- recall: OWL Variants:
  - OWL Lite (not explicitly discussed on the previous slides ...),
  - OWL-DL (equivalent to Description Logics),
  - OWL Full (syntax of RDF+RDFS+OWL, semantics undecidable, partially critical)
- From practical considerations, the *OWL profiles* are more important:
  `http://www.w3.org/TR/owl2-profiles/`
  - OWL2-EL: for ontologies that contain very large numbers of properties and/or classes. Basic reasoning problems can be performed in time that is polynomial *with respect to the size of the ontology* – not to the data
  - OWL2-QL: applications that use very large volumes of instance data, and where query answering is the most important reasoning task. Conjunctive query answering can be implemented using conventional relational database systems.
  - OWL-RL: scalable reasoning without sacrificing too much expressive power. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines.
  - note: none of the profiles is a subset of another

OWL2-RL

```
https://www.w3.org/TR/owl2-profiles/#OWL_2_RL
```

- OWL2-RL constrains the set of allowed constructs (e.g., no cardinalities other than 0,1,*), *and the usage* of constructs.
- An important restriction is the usage in subClassOf axioms:
  - Consider that subclass assertions are often used in RDFS:
    $\exists p.D \sqsubseteq C$ (domain) and $C \sqsubseteq \forall p.D$ (range).
    Typical OWL usage:
    $C \sqsubseteq \exists p.D$ to assert existential things, needs skolemization, can create infinite chains,
    $\forall p.D \sqsubseteq C$ check whether a universal restriction holds to define a class; needs to prove non-existence
  - OWL2-RL restriction:
    - $\exists p.D$ only on the subclass side,
    - $\forall p.D$ only on the superclass side,
    - compatible with RDFS, not with more expressive OWL use.

OWL2-RL (cont'd)

- Usage of Complement: only in superclass position *classexpr* $\sqsubseteq \neg D \rightarrow$ usage only as constraint to state what is not allowed
- Usage of Union and owl:oneOf only in subclass position:
  $(C_1 \sqcup C_2) \sqsubseteq D \quad \rightarrow \quad C_1 \sqsubseteq D$ and $C_2 \sqsubseteq D$, not in the head as true choice.
- usage of maxCardinality only 0,1 and only on superclass side (as restrictions)
- intersection and $\exists p.obj$ and $\exists p.value$ is allowed on both sides

⇒ does not cover everything of OWL Lite (= $\mathcal{SHIF}$, allows $\exists p.D$ without restriction) but also covers parts of OWL DL.

Translation to rules see `https://www.w3.org/TR/owl2-profiles/#OWL_2_RL`
(Entries prp-npa1, prp-npa2 seem to be wrong (require (?x a NegativeObject/DataPropertyAssertion)). For cls-svf-1 and csl-svf-2 consider that $\exists p.C$ is only allowed as subclass axiom $\exists p.C \sqsubseteq D$.)

- The internal rule-based reasoner of Jena goes beyond OWL-RL (and can run into nontermination, see Slide 617)

RULE-BASED REASONING

- cf. Datalog
- first-order: cannot reason about classes (predicate symbols), but only about individuals
- might apply second-order rule *patterns* e.g. for transitivity, subclass/subproperty, domain/range ("syntactically second-order", but actually only a first-order mechanism)
- OWL world: open-world negation, Rules: closed-world negation
  - ⇒ no way out in this issue (recall that well-founded semantics and stable models are also CWA)
  - ⇒ allow only positive rules
- For rule equivalents to the OWL-RL constructs allowed on OWL-RL see `https://www.w3.org/TR/owl2-profiles/#OWL_2_RL`.
  Choose those that are needed for a certain problem.
- Rule-based semantics has problems when allowing to derive existential objects: in OWL-RL *class [membership] assertions* (c a *class*), classes of the type $\exists p.C$ are not allowed.
- user-defined rules can be added.

11.1 Reasoning in Jena

`https://jena.apache.org/documentation/inference/`

Using an Ontology-Aware Reasoner

- can be used with the OntModel interface
- OntModel provides query/update methods on the level of Classes/Properties/DL-Concepts
- initialize the Model with a certain "OntModelSpec" (= complexity/expressiveness level) where Jena automatically employs and configures an internal OWL reasoner.
- can also be initialized with "foreign" reasoners that provide Jena adaptation (e.g. Pellet/Openllet)

Using another Reasoner

- can be used with the InfModel interface
- OntModel is a "subinterface" of InfModel, i.e. OntModel refines/extends InfModel
- combines a (=any) reasoner with an RDF model
  - used for "alien" external reasoners
  - used for Jena's (non-ontology-aware) General Rule-Based Reasoner

Using Pellet as Reasoner

- Pellet is a tableau-based OWL-DL reasoner which can be used with an OntModel.

Example: Win-Move-Game: Draw Nodes

- Pure OWL cannot populate the DrawNode class
- use procedural programming to add triples (x a :DrawNode) to the model after reasoning
- note: it is not necessary to call prepareModel() explicitly
- Java Code next slide

```
import org.apache.jena.util.FileManager;
import openllet.jena.PelletReasonerFactory;

public class JenaPelletWinMoveDraw {
  static String filepath = "/home/may/teaching/SemWeb/RDF/";
  public static void main(String[] args){
    Model m = FileManager.get().loadModel(filepath + "winmove-axioms.n3");
    m.add(FileManager.get().loadModel(filepath + "winmove-closure.n3"));
    m.add(FileManager.get().loadModel(filepath + "winmove-graph.n3"));
    OntModel pelletmodel = ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC, m);

    String q = "prefix : <foo://bla#>" +
        "construct { ?N a :DrawNode }" +
        "where { ?N a :Node . filter not exists { ?N a :WinNode }" +
        "                      filter not exists { ?N a :LoseNode }}";
    Query qu = QueryFactory.create(q);
    QueryExecution qe = QueryExecutionFactory.create(qu, pelletmodel);
    Model resultgraph = qe.execConstruct();

    pelletmodel.add(resultgraph);

    q = "prefix : <foo://bla#>" +
        "select ?W ?L ?D " +
        "where {{ ?W a :WinNode } union { ?L a :LoseNode } union { ?D a :DrawNode }}";
    qu = QueryFactory.create(q);
    qe = QueryExecutionFactory.create(qu, pelletmodel);
    ResultSet results = qe.execSelect();
    ResultSetFormatter.out(results);
} }
```

[Filename: Java/JenaPelletWinMoveDraw.java]

Insights into derivations in Jena's InfModel (and OntModel)

- `Model getRawModel()`: the underlying RDF graph only.

Further methods are only partially (dependent on the reasoner) supported:

- `Model getDeductionsModel()`: The triples that are added to the base graph due to reasoning.

  Before this operation, all reasoning is evaluated first.
  It is not possible to see an "intermediate", partially on-demand state.

- if getDeductionsModel() does not correctly return all deductions:
  model.difference(model.getRawModel()).difference(model.getDeductionsModel())
  yields derived statements that are *not* in the Deductions Model ...

- `void setDerivationLogging(boolean logOn)`

  Switch on/off drivation logging. This can use a lot of space.

- `Iterator<Derivation> getDerivation(Statement stmt)`

- Derivation:
  - `printTrace(PrintWriter out, boolean bindings)`

- next slide: Java test code with pellet.

```
public class JenaPelletWinMove {
  static String filepath = "/home/may/teaching/SemWeb/RDF/";
  public static void main(String[] args){
    Model m = FileManager.get().loadModel(filepath + "winmove-axioms.n3");
    m.add(FileManager.get().loadModel(filepath + "winmove-closure.n3"));
    m.add(FileManager.get().loadModel(filepath + "winmove-graph.n3"));
    OntModel pelletmodel = ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC, m);
    Reasoner pellet = pelletmodel.getReasoner();
    pellet.setDerivationLogging(true);
    pelletmodel.prepare();

    Model dedModel = pelletmodel.getDeductionsModel();
    if (dedModel == null) System.out.println("DedModel is null");   // it is null ...
        else dedModel.write(System.out,"N3");
    Model rawModel = pelletmodel.getRawModel();
    Model diffModel = pelletmodel.difference(rawModel); // .difference(dedModel);
    System.out.println("---- DiffModel: ----------------------");
    diffModel.write(System.out,"N3");   // everything is in the DiffModel

    PrintWriter out = new PrintWriter(System.out);  // no derivations available
    for (StmtIterator i = diffModel.listStatements(null, RDF.type, (RDFNode) null); i.hasNext(); ) {
        Statement s = i.nextStatement();
        out.write("Statement is " + s + "\n");
        Iterator <Derivation> ds = pelletmodel.getDerivation(s);
        if (ds == null) System.out.println("DerivationsIterator is null");
        else for (Iterator<Derivation> i2 = pelletmodel.getDerivation(s); i2.hasNext(); ) {
            Derivation deriv = (Derivation) i2.next();
            deriv.printTrace(out, true);
    }   }
    out.flush();  } }
```

[Filename: Java/JenaPelletWinMove.java]

Code/Test Example: Pellet, Win-Move-Game

- getDeductionsModel() is null
- conclusions are in diffModel = pelletmodel.difference(rawModel);
- no derivation information available
  - Derivation information is especially useful, when an ontology is derived to be inconsistent
  - ⇒ pellet returns error messages that do not always give good insights where the problem is located
  - the DL-Prover “Hermit” (DL only, no SPARQL) provides better functionality for ontology management.

11.1.1 The Rule-Based OWL Reasoner in Jena

- Jena’s built-in `OntModelSpec`s activate the internal rule-based engine with corresponding specifications
- the most powerful one is OWL_DL_MEM_RULE_INF
  `https://jena.apache.org/documentation/inference/#owl`
- Extends the RDFS reasoner,
  “since RDFS is not a subset of the OWL/Lite or OWL/DL languages the Jena implementation is an incomplete implementation of OWL/full”
  ... means that OWL Full is an upper bound of it, but it does eben not (really) *cover* OWL-DL, nor OWL-Lite, nor OWL-RL.
- call
  ```
  jena -inf -if inputfile -qf queryfile
  ```
- also available in the Web interface

The Rule-Based OWL Reasoner

- Strategy: an instance-based reasoner
- Reasoning about class hierarchy: prove things like "$C_1 \sqsubseteq C_2$":

  add a "prototypical individual" x_{C_1} (which does not have any other properties) to C_1, apply all rules (model completion), and check whether $C_2(x_{C_1})$ is concluded.
  (These individuals should not longer be visible in user queries since Jena 2.1)
- "The OWL_Mini reasoner ... omits the forward entailments from minCardinality/ someValuesFrom restrictions - that is it avoids introducing bNodes which avoids some infinite expansions"

  ... that tells how to trap the "full" OWL_RULE reasoner with "parents"
  (cf. Slide 620)
  - Tries to provide more than OWL-RL (which excludes SomeValuesFrom specifications because they introduce implicit objects, which requires a blocking algorithm)
  - even OWL Lite includes SomeValuesFrom, even with functionality restriction ($\mathcal{SHIF}$).

The Rule-Based OWL Reasoner (cont'd)

- OWL_Micro reasoner: "RDFS plus the various property axioms, intersectionOf, unionOf (partial) and hasValue. It omits the cardinality restrictions and equality axioms, which enables it to achieve much higher performance."
- it seems that the OWL_RULE reasoner also ignores cardinalities, including "minCardinality 1" (example: (parents, 2), and also (parents, 1)).
- "The critical constructs which go beyond OWL/lite and are not supported in the Jena OWL reasoner are complementOf and oneOf. As noted above the support for unionOf is partial (due to limitations of the rule based approach) but is useful for traversing class hierarchies."
  $\Rightarrow$ No negation, very restricted disjunction (not as choice, only as union), no case-based reasoning

  (cf. the parricides, the meals-wine-ontology from Deductive Databases [Exercise], and win-move)
- "Even within these constructs rule based implementations are limited in the extent to which they can handle equality reasoning - propositions provable by reasoning over concrete and introduced instances are covered but reasoning by cases is not supported."

The Rule-Based OWL Reasoner (cont'd)

Note: Datalog with well-founded semantics, which is polynomial, solves win-move (but not the others).

- the OWL_RULE-Reasoner does not support cardinalities that are necessary to close the "edge" relation.
- even with this, testing membership in AllValuesFrom(edge, LoseNode) would not be possible/easy: [if find-n-edges-to-"lose" then "win"] could be encoded in a set of rules, but the well-founded evaluation is hard.

SomeValuesFrom – Trapping the OWL Reasoner

Recall the example for the blocking strategy (Slide 519): every person has two parents, which are again persons.

- `jena -q -pellet -qf infinite-parents.sparql` ... pellet correctly does the blocking,
- `jena -q -inf -qf infinite-parents.sparql` ... ... runs forever.
- Aside: when deleting the last 2 lines from `infinite-parents.sparql` ($\mathsf{HasParent} \equiv \exists \mathsf{hasParent}.\top$), the computation finishes with the same output as pellet (and some blanknodes).
  The still existing $\mathsf{Person} \sqsubseteq \exists^{=2}\mathsf{hasParent}.\top$ is ignored. Replacing this by $\mathsf{Person} \sqsubseteq \exists^{=1}\mathsf{hasParent}.\top$ (cardinality 1) leads to some more blank nodes, but $\mathsf{Person} \sqsubseteq \exists \mathsf{hasParent}.\mathsf{Person}$ (someValuesFrom Person) again fails.

The OWL_Mini Reasoner: ignores SomeValuesFrom

- With the OWL_Mini Reasoner, the example runs, but the SomeValuesFrom is completely ignored: Nobody is a HasParent (because then, its definition is empty).
  Java code see next slide.

```
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;

public class JenaIntRuleMicro {
  static String filepath = "/home/may/teaching/SemWeb/RDF/";
  public static void main(String[] args){
    Model m = FileManager.get().loadModel(filepath + "infinite-parents.n3");
    OntModel ontmodel = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF, m);
    // OntModel ontmodel =
    //   ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_MICRO_RULE_INF, m);
    ontmodel.prepare();
    System.out.println(" ... prepared the model ...");

    String q = "prefix : <foo://bla#> " +
        "select ?A ?C ?X " +
        "where {{?A a :Parent} UNION {?C a :HasParent} UNION {:kate :parent ?X}}";

    Query qu = QueryFactory.create(q);
    QueryExecution qe = QueryExecutionFactory.create(qu, ontmodel);
    ResultSet results = qe.execSelect();
    ResultSetFormatter.out(results);
} }
```

[Filename: Java/JenaIntRuleMicro.java]

Code/Test Example: Deduction tracing in the Jena Rule-Based OWL Reasoner

- next slide: Java test example for tracing deductions (transitivity: descendants)
- getDeductionsModel() contains lots of trivial OWL axioms.
- useful conclusions (and lots of other conclusions) are in diffModel = ont.difference(rawModel).difference(dedModel);
- no derivation information available.

Further evaluation of its functionality: Exercise.

```
  static String filepath = "/home/may/teaching/SemWeb/RDF/";
  public static void main(String[] args){
    Model m = FileManager.get().loadModel(filepath + "descendants.n3");
    OntModel ontmodel = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM_RULE_INF, m);
    Reasoner reasoner = ontmodel.getReasoner();
    reasoner.setDerivationLogging(true);

    Model dedModel = ontmodel.getDeductionsModel();
    if (dedModel == null) System.out.println("DedModel is null");  // it is null ...
        else dedModel.write(System.out,"N3");   // viele triviale OWL-Axiome im DedModel
    Model rawModel = ontmodel.getRawModel();
    Model diffModel = ontmodel.difference(rawModel).difference(dedModel);
    System.out.println("---- DiffModel: ----------------------");
    diffModel.write(System.out,"N3");  // vieles, incl die sinnvollen Ergebnisse im DiffModel

    PrintWriter out = new PrintWriter(System.out);
    for (StmtIterator i = diffModel.listStatements(null,
        ontmodel.getProperty("foo://bla/meta#descendant"),
        (RDFNode) null); i.hasNext(); ) {
        Statement s = i.nextStatement();
        out.write("Statement is " + s + "\n");
        Iterator <Derivation> ds = ontmodel.getDerivation(s);
        if (ds == null) System.out.println("DerivationsIterator is null");
        else for (Iterator<Derivation> i2 = ontmodel.getDerivation(s); i2.hasNext(); ) {
            Derivation deriv = (Derivation) i2.next();
            deriv.printTrace(out, true);   // derivations are not null, but empty
    }   }
    out.flush();
}}
```
[Filename: Java/JenaIntRuleDeductions.java]

11.1.2 The Generic Rule-Based Reasoner in Jena

- allows three types of rule handling (cf. Lecture “Deductive Databases”):
  - forwards-chaining bottom-up (T_P^ω-style; efficient well-known RETE algorithm)
    - if a *ground instance of the rule body matches facts in the DB*, the instance of the head atom is derived to extend the RDF graph with additional derived facts.
    - with *hybrid rules* (cf. Slide 634; rules whose head is a (backward) rule): derive additional rules that extend the program
  - backwards-chaining top-down (Prolog-SLG resolution, basically like XSB including tabling etc.)
    - if the *rule head matches a query*, it is tried to find answers for the body (whose atoms may be basic predicates or match the heads of other rules)
  - hybrid: backward-and-forward mixed. The user can for each rule define how it should be interpreted. (default configuration)

Call with semweb.jar - command line arguments

- -if, -q, -qf as usual
- -rf *rulefile*
- -fw, -bw, -bwfw or -hybrid

Basic Rule Syntax: File or String with ...

- `https://jena.apache.org/documentation/inference/#RULEsyntax`
- rules as "[*head* <- *body*]" or "[*body* -> *head*]" ,
  or "*head* <- *body* ." or "*body* -> *head* ." (as in Datalog),
  - -fw reasoner interprets only "->" rules, multiple head atoms are allowed (left-to-right exec),
  - -bw reasoner interprets all rules (handle then as backward rules, Prolog-style),
  - -hybrid interprets both types in 2 stages: fw first, then afterwards bw
- triple patterns as "(s p o)" with variables ?x as in SPARQL; equal(?x,?y)
- "?x a ?c" is not understood, use "?x rdf:type ?c" instead
  (note: using "a" does not cause an error message, but simply nothing is matched)
- body: lessThan(?x,?y), sum(?a,?b,?c) (if safe, cf. Deductive Databases lecture), now(),
- only in forward rules:
  - body: constructors like strConcat(...,?res), uriConcat(...,?res), makeSkolem(...)
  - updates like remove(n) in the head (removes ground instance of the n-th body term),
- handling/iterating over RDF lists,
- no aggregation (?)

Rules Example

```
@prefix : <http://www.semwebtech.org/mondial/10/meta#>.
[ (?x rdf:type :Bigcountry)
   <-
  (?x rdf:type :Country), (?x :population ?p), greaterThan(?p, 10000000) ]
```

[Filename: RDF/rule-bigcountries-bw.rl]

```
@prefix : <http://www.semwebtech.org/mondial/10/meta#>.
[ (?x rdf:type :Country), (?x :population ?p), greaterThan(?p, 10000000)
   ->
  (?x rdf:type :Bigcountry) ]
```

[Filename: RDF/rule-bigcountries-fw.rl]

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?C ?N
# from <file:mondial-europe.n3>
where { ?C a :Country .
        optional {?C a :Bigcountry; :name ?N}}
```

[Filename: RDF/rule-bigcountries.sparql]

```
jena -if mondial-europe.n3 -rf rule-bigcountries-bw.rl \
     -qf rule-bigcountries.sparql -bw
```

Rules Example

```
@prefix : <http://www.semwebtech.org/mondial/10/meta#>
[ (?x rdf:type :Country), (?x :population ?p), greaterThan(?p, 10000000),
  strConcat('A','B',?res)    ### to test strConcat in fwd/bwd eval
    -> (?x rdf:type :Bigcountry), (?x :testprop ?res) ]
[ (?x :bigneighborwith ?y)
  <- (?x rdf:type :Bigcountry), (?x :neighbor ?y), (?y rdf:type :Bigcountry) ]
```

[Filename: RDF/rule-neighbor-bigcountries.rl]

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?N ?X ?TT
# from <file:mondial-europe.n3>
where {{ <http://www.semwebtech.org/mondial/10/countries/D/>
         :bigneighborwith ?X }
       union { ?C a :Bigcountry; :name ?N OPTIONAL { ?C :testprop ?TT}}
      }
```

[Filename: RDF/rule-neighbor-bigcountries.sparql]

- -bwfw/hybrid fill both (also when union line commented out) and sets :testprop
- -fw fills only “Bigcountry” (?N) (i.e., ignores backward rules)
- -bw fills both, but ignores the strConcat literal in the fwd rule, does not fill :textprop

Rules Example

- same rule, other directions:

```
@prefix : <http://www.semwebtech.org/mondial/10/meta#>
[ (?x rdf:type :Bigcountry)
  <- (?x rdf:type :Country) , (?x :population ?p), greaterThan(?p, 10000000) ]

[ (?x rdf:type :Bigcountry), (?x :neighbor ?y), (?y rdf:type :Bigcountry)
  -> (?x :bigneighborwith ?y)  ]
```

[Filename: RDF/rule-neighbor-bigcountries2.rl]

- -bw fills both (i.e. also evaluates forward rules, then in backward direction)
- -bwfw fills only “Bigcountry” (?N)
- -fw fills nothing, (ignores backward rules, and the “lower” rule is backward)

Reasoners' Behavior

- Leveling of the hybrid reasoner:
  - forward rules first (filling "views")
  - backwards rules afterwards
- Backward-chaining reasoner also evaluates forward rules, then in backward direction,
- Forward-chaining reasoner ignores backward rules.

Including other rule sets

- `@include` <*otherrulefile*> in the rule file.
- instead of <*otherrulefile*>, also keywords RDFS, OWL, OWLMini, and OWLMicro are allowed to preload the respective rule sets.

Loading Rules in Java

- List<Rule> rules = Rule.rulesFromURL(*http:-URL or file*);
- String ruleSrc = "*list of rules as string*"
  List rules = Rule.parseRules(rulesSrc);

Tabling

- Analogously to Prolog/Datalog: In top-down/backward evaluation, intermediate results are cached ("tabled").
  This is especially necessary for stratified negation, but also more efficient whenever some subgoal can be reused.
- tableAll(), table(P). If any property is tabled, goals such as (?A, ?P, ?X) will all be tabled because the property variable might match one of the tabled properties.
- Syntax in rule file: `[ -> tableAll() ]` or `[ -> table(:bigneighborwith) ]`
- The tabled results of each query are kept indefinitely. Queries can exploit all of the results of the subgoals involved in previous queries. In essence we build up a closure of the data set in response to successive queries.
- tabling: reset()
- When the inference model is updated by adding or removing statements all tabled results are discarded by an internal reset() and the next query will rebuild the tabled results from scratch.
- Java Code next slides. Tabling of backward reasoning is not stored in the DeductionModel.

```
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;

public class JenaRules {
  static String filepath = "/home/may/teaching/SemWeb/RDF/";
  public static void main(String[] args){
    // Model model = ModelFactory.createDefaultModel();
    Model m = FileManager.get().loadModel(filepath + "mondial-europe.n3");
    List<Rule> rules = Rule.rulesFromURL(filepath + "rule-neighbor-bigcountries.rl");
    GenericRuleReasoner reasoner = new GenericRuleReasoner(rules);
    reasoner.setMode(GenericRuleReasoner.HYBRID);
    InfModel model = ModelFactory.createInfModel(reasoner, m);
    String q = "prefix mon: <http://www.semwebtech.org/mondial/10/meta#>" +
            "select ?N ?X" +
              " where {{ ?Z" + // <http://www.semwebtech.org/mondial/10/countries/D/>" +
            "          mon:bigneighborwith ?X }" +
            " union { ?C a mon:Bigcountry; mon:name ?N }}";
    Query qu = QueryFactory.create(q);
    QueryExecution qe = QueryExecutionFactory.create(qu, model);
    ResultSet results = qe.execSelect();
    ResultSetFormatter.out(System.out, results, qu);

    model.getDeductionsModel().write(System.out,"N3");
  }
}
```

[Filename: Java/JenaRules.java]

```
    GenericRuleReasoner reasoner = new GenericRuleReasoner(rules);
    reasoner.setMode(GenericRuleReasoner.HYBRID);
    reasoner.setDerivationLogging(true);
    InfModel model = ModelFactory.createInfModel(reasoner, m);
    String q = "prefix mon: <http://www.semwebtech.org/mondial/10/meta#>" +
            "select ?N ?X" +
              " where {{ <http://www.semwebtech.org/mondial/10/countries/D/>" +
            "            mon:bigneighborwith ?X }" +
            "  union { ?C a mon:Bigcountry; mon:name ?N }}";
    Query qu = QueryFactory.create(q);
    QueryExecution qe = QueryExecutionFactory.create(qu, model);
    ResultSet results = qe.execSelect();
    ResultSetFormatter.out(System.out, results, qu);

    Model dedModel = model.getDeductionsModel();
    Model rawModel = model.getRawModel();
    Model diffModel = model.difference(rawModel).difference(dedModel); // = bwd derivations
    System.out.println("---- Derivations (in the DedModel) : ----------------------");
    dedModel.write(System.out,"N3");
    System.out.println("---- Derivations (in the DiffModel) : ----------------------");
    PrintWriter out = new PrintWriter(System.out);
    for (StmtIterator i = diffModel.listStatements(null,
          model.getProperty("http://www.semwebtech.org/mondial/10/meta#bigneighborwith"),
          (RDFNode) null); i.hasNext(); ) {
        Statement s = i.nextStatement();
        out.write("Statement is " + s + "\n");
        for (Iterator<Derivation> i2 = model.getDerivation(s); i2.hasNext(); ) {
            Derivation deriv = (Derivation) i2.next();
            deriv.printTrace(out, true);
        } }
    out.flush(); } }
```

[Filename: Java/JenaRulesDedModel.java]

Insights into derivations with the GenericRuleReasoner (Hybrid)

- `Model getDeductionsModel():`
  - even if no query is stated, the *full deductive closure inc. backward chaining rules is evaluated* when executing such model operations.
  - getDeductionsModel() contains only for forward rule firings. This allows the forward rules to be used separately as if they were rewrite transformation rules (create *new* graph *without* old).
  - Facts derived by the backward-chaining rules: intermediate results are tabled (internally), and added to the InfModel, but neither in the rawModel nor in the deductionsModel.
  - For all derived facts, derivation trees (their T_P^ω-derivation) are available.

Forward rules fire on ontology updates

- With the forward chaining reasoner, if the InfModel is changed (add or remove triples) through the API, this triggers rule evaluations incrementally (RETE algorithm).

HYBRID RULES: GENERATION OF NEW RULES BY FORWARD RULES GENERATING RULES

- Forward rules are allowed to create new rules (only backward rules) in their *heads*. (Then, use the “[...]” syntax for rules to allow for nesting)
- mostly used to break down second-order patterns to first order instantiations (like e.g., the transitivity pattern):

Example 1

Consider the case of rdfs:subPropertyOf assertions:

```
  [   (?a ?q ?b) <- (?p rdfs:subPropertyOf ?q), (?a ?p ?b) .]
```

as a backward rule. For *every* (sub)goal of the form (?x anyprop ?v), the head will match, and the subgoal is replaced by (?p rdfs:subPropertyOf anyprop), (?x ?p ?v), usually only for finding out that *anyprop* does not have subproperties.

Thus, the application cases can be restricted as follows: Add a *hybrid rule* whose outer, forward rule, “fires” for each (p rdfs:subProperty q) fact and creates a *partial instance* of the above rule:

```
[ (?p rdfs:subPropertyOf ?q), notEqual(?p,?q) -> [ (?a ?q ?b) <- (?a ?p ?b) ] ]
```

Thus, for, e.g., a statement (:cityIn rdfs:subPropertyOf :locatedIn) it will add the rule

```
[  (?a :locatedIn ?b) <- (?a :cityIn ?b) ]
```

which matches only subgoals of the form (?x :locatedIn ?c).

Example 2: Transitivity

Transitivity is a typical “syntactically second-order propery” which can be mapped to its first-order instantiated pattern. – Exercise.

Example 3: Property Chains

The “Property Chain/Role Chain” pattern is another example for such a hybrid rule:

- the owl:PropertyChain construct (cf. Slide 502) is internally (and in RDF/XML, see Slide 503) mapped to an RDF list, but this is immediately “consumed” by the parser and appropriate knowledge is added to the model.

```
[ owl:propertyChain (:brotherOf :hasChild) ]
  rdfs:subPropertyOf :uncleOf.
```

- For an independent rule-based handling, use another suitable presentation by RDF triples that connect the chain property with its constituents.
  Note that these triples connect properties, so they are not common RDF properties, but must be declared as owl:AnnotationProperties, which are ignored by OWL reasoning, but nevertheless accessible for graph-level queries with SPARQL and in rule bodies:

```
:propchainFirst a owl:AnnotationProperty.
:propchainSecond a owl:AnnotationProperty.
:hasUncle :propchainFirst :hasParent ; :propchainSecond :hasSibling .
```

- The forward rule pattern must then match the above and generate appropriately instantiated backward rules, here

```
[ (?X :uncleOf ?Z) <- (?X :brotherOf ?Y, ?Y :hasChild ?Z) ]
```

Example: Property Chains (cont'd)

```
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix : <foo://bla#>.
  :john :hasChild :bob; :hasSibling :paul, [].
  :hasSibling a owl:SymmetricProperty.
  :paul a [a owl:Restriction; owl:onProperty :hasChild; owl:minCardinality 1].
  :hasChild owl:inverseOf :hasParent.
:propchainFirst a owl:AnnotationProperty.
:propchainSecond a owl:AnnotationProperty.
:hasUncle :propchainFirst :hasParent ; :propchainSecond :hasSibling .
```

[Filename: RDF/uncle-rule-rl.n3]

```
@prefix : <foo://bla#>.
@include <OWL>.
[ (?r :propchainFirst ?p), (?r :propchainSecond ?q) ->
         [ (?x ?r ?y) <- (?x ?p ?z) (?z ?q ?y)] ]
```

[Filename: RDF/uncle-rule-rl.rl]

```
prefix : <foo://bla#>
select ?P ?U
where { ?P :hasUncle ?U }
```

[Filename: RDF/uncle-rule-rl.sparql]

call: jena -hybrid -if uncle-rule-rl.n3 -rf uncle-rule-rl.rl -qf uncle-rule-rl.sparql

Functors in Rules [additional syntax, ignore]

Functor syntax within rules does allow creation of nested "data structures": $f(a, b, c, d)$

- Datalog: allows arbitrary n-ary predicate symbols in rule heads (EDB)
- RDF: allows only triples,
  complex relationships cannot be described in a single atom,
  $\rightarrow$ requires several triples for reification
- Functors: *auxiliary syntactical sugar* allowed to create in heads of *forward rules* and in rule bodies.

$\Rightarrow$ such terms are internal to the rule evaluation,
finally there must be a rule that creates triple instances into the graph.

- functor terms may be nested.

Generic Rule Reasoner + OWL/RDFS rules

`https://jena.apache.org/documentation/inference/#RDFSPlusRules`

- use the GenericRuleReasoner to combine rules for RDFS or OWL and user-defined rules:

```
reasoner.setOWLTranslation(true);
reasoner.setTransitiveClosureCaching(true);
```

- default Jena RDFS and OWL rulesets use the Hybrid rule engine. The hybrid engine is itself layered, forward rules do not see the results of any backward rules.
- all inferences that must be seen *by the RDF/OWL rules* must be forward, all the inferences which need the results of the RDFS/OWL rules must be backward.
  - "complete" RDF graph by forward-chaining rules (like uncleOf etc.)
  - then fwd-bwd predefined OWL rules,
  - further backward-chaining rules building upon OWL conclusions.

Conclusion

- Rules cannot have negative literals in the body

⇒ the rules implementing the OWL fragment can also not use negation
(OWL has open world, rules have closed world)

- ... not more than positive Datalog restricted to triples??

Chapter 12
Conclusion and Outlook

What should have been learnt:

- Formal Logic: interpretations, model theory, first-order logic
- Deductive systems: Datalog, minimal model semantics
- reasoning: tableau calculi
- RDF as a special, simple data model; URIs
  representations: Turtle and RDF/XML
- DL as another logic, Open World
- "database" vs. "knowledge base"
- OWL as "DL alive"

Semantic Web Data: XML; RDF and OWL

In contrast to XPath/XQuery, XSLT, XML Schema, XLink etc., RDF and OWL are *not* languages *“inside”* the XML world, but are concepts of their own that have - incidentally- also an XML syntax.

The combination of XML data and RDF/RDFS/OWL concepts is the base for the *Semantic Web*.

A Semantic Web application e.g. exists of

- a “central” portal that uses the following things:
- a set of ontological (OWL, RDFS) sources,
- a set of RDF sources,
- reasoning (using OWL and RDFS information),
- a semantical description of itself for allowing others to use it.