OWL 2
Web Ontology Language
Revised

Some material adapted from presentations by Ian Horrocks and by Feroz Farazi
OWL 1

• TASK: What do you recall about OWL 1?
  – Constructors
  – Language Fragments
  – Major problem?
OWL: DLs as foundation

- OWL DL based on SHIQ Description Logic
  - In fact it is equivalent to $\text{SHOIN(D}_n\text{)}$ DL

- OWL DL Benefits from many years of DL research
  - Well defined semantics
  - Formal properties well understood (complexity, decidability)
  - Known reasoning algorithms
  - Implemented systems (highly optimised)

- In fact there are three “species” of OWL (!)
  - OWL full is union of OWL syntax and RDF
  - OWL DL restricted to First Order fragment ($\frac{1}{4}$ DAML+OIL)
  - OWL Lite is “simpler” subset of OWL DL (equiv to $\text{SHIF(D}_n\text{)}$)
# Class/Concept Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
<th>FOL Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>$C_1 \sqcap \ldots \sqcap C_n$</td>
<td>Human $\sqcap$ Male</td>
<td>$C_1(x) \land \ldots \land C_n(x)$</td>
</tr>
<tr>
<td>unionOf</td>
<td>$C_1 \sqcup \ldots \sqcup C_n$</td>
<td>Doctor $\sqcup$ Lawyer</td>
<td>$C_1(x) \lor \ldots \lor C_n(x)$</td>
</tr>
<tr>
<td>complementOf</td>
<td>$\neg C$</td>
<td>$\neg$ Male</td>
<td>$\neg C(x)$</td>
</tr>
<tr>
<td>oneOf</td>
<td>${x_1} \sqcup \ldots \sqcup {x_n}$</td>
<td>${john} \sqcup {mary}$</td>
<td>$x = x_1 \lor \ldots \lor x = x_n$</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>$\forall P.C$</td>
<td>$\forall$ hasChild.Doctor</td>
<td>$\forall y.P(x,y) \rightarrow C(y)$</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>$\exists P.C$</td>
<td>$\exists$ hasChild.Lawyer</td>
<td>$\exists y.P(x,y) \land C(y)$</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>$\leq n P$</td>
<td>$\leq 1$ hasChild</td>
<td>$\exists y.P(x,y)$</td>
</tr>
<tr>
<td>minCardinality</td>
<td>$\geq n P$</td>
<td>$\geq 2$ hasChild</td>
<td>$\exists \geq n y.P(x,y)$</td>
</tr>
</tbody>
</table>

- $C$ is a concept (class); $P$ is a role (property); $x$ is an individual name
Turtle Syntax

• :HappyFather type owl:Class;
  Owl:equivalentClass
  [
    type owl:Restriction;
    owl:onProperty :hasChild;
    owl:allValuesFrom :Happy
  ]
OWL 2

• OWL 2 extends OWL 1 and is backward compatible with it
• The new features of OWL 2 based on real applications, use cases and user experience
• Adopted as a W3C recommendation in December 2009
• All new features were justified by use cases and examples
Features and Rationale

• Syntactic sugar
• New constructs for properties
• Extended datatypes
• Punning
• Extended annotations
• Some innovations
• Minor features
Syntactic Sugar

• OWL 2 adds features that
  – Don’t change expressiveness, semantics, complexity
  – Makes some patterns easier to write
  – Allowing more efficient processing in reasoners

• New features include:
  – DisjointUnion
  – DisjointClasses
  – NegativeObjectPropertyAssertion
  – NegativeDataPropertyAssertion
Syntactic sugar: disJointUnion

• Need for disjointUnion construct
  – A :CarDoor is exclusively either
    • a :FrontDoor,
    • a :RearDoor or
    • a :TrunkDoor
    • and not more than one of them

• TASK:
  • How to do that in OWL 1?
Syntactic sugar: disJointUnion

• Need for disjointUnion construct
  – A :CarDoor is exclusively either
    • a :FrontDoor,
    • a :RearDoor or
    • a :TrunkDoor
    • and not more than one of them

• In turtle (OWL 2)
  :CarDoor a owl:Class;
  owl:disjointUnionOf
    (:FrontDoor
     :RearDoor
     :TrunkDoor) .
Syntactic sugar: disJointUnion

• It’s common for a concept to have more than one decomposition into disjoint union sets
• E.g.: every person is either male or female (but not both) and also either a minor or adult (but not both)

    foaf:Person
    owl:disjointUnionOf (:MalePerson :FemalePerson);
    owl:disjointUnionOf (:Minor :Adult) .
Syntactic sugar: disJointClasses

• It’s common to want to assert that a set of classes are pairwise disjoint
• i.e., that no individual can be an instance of two of the classes in the set
• Task: How to do that in OWL 1?
Syntactic sugar: disJointClasses

• It’s common to want to assert that a set of classes are pairwise disjoint
• i.e., that no individual can be an instance of two of the classes in the set
• OWL 2:
  [a owl:allDisjointClasses;
   owl:members (:faculty :staff :students)]
Syntactic sugar: negative assertions

• Asserts that a property doesn’t hold between two instances or between an instance and a literal
  • NegativeObjectPropertyAssertion
    – Barack Obama was not born in Kenya
  • NegativeDataPropertyAssertion
    – Barack Obama is not 60 years old
• Encoded using a “reification style”
Syntactict sugar: negative assertions

@prefix dbp: <http://dbpedia.org/resource/> .
@prefix dbpo: <http://dbpedia.org/ontology/> .

[a owl:NegativeObjectPropertyAssertion;
 owl:sourceIndividual dbp:Barack_Obama ;
 owl:assertionProperty dbpo:born_in ;

[a owl:NegativeDataPropertyAssertion;
 owl:sourceIndividual dbp:Barack_Obama ;
 owl:assertionProperty dbpo:age ;
 owl:targetIndividual "60" ] .
New property Features

• Self restriction
• Qualified cardinality restriction
• Object properties
• Disjoint properties
• Property chain
• keys
Self restriction

• Classes of objects that are related to themselves by a given property

• For example, the class of processes that regulate themselves

• It is also called local reflexivity
  – narcissists are people who love themselves
Self restriction

• TASK:
  – Think about how to adapt the tableaux rules to cover concept description using self.
Qualified cardinality restrictions

• Qualifies the instances to be counted
• Six varieties: \{Data | Object\}\{Min | Exact | Max\}Cardinality
• For example,
  – People with exactly three children who are girls
  – People with at least three names
  – Each individual has at most one SSN
Qualified cardinality restrictions

• Done via new properties with domain owl:Restriction, namely \{min\|max\}\)QualifiedCardinality and onClass

• Example: people with exactly three children who are girls

  [a owl:restriction;
   owl:onProperty :has_child;
   owl:onClass [owl:subClassOf :FemalePerson;
               owl:subClassOf :Minor].

  QualifiedCardinality “3” .
Object properties

• ReflexiveObjectProperty
  – Globally reflexive
  – Everything is part of itself

• IrreflexiveObjectProperty
  – Nothing can be a proper part of itself

• AsymmetricObjectProperty
  – If \( x \) is proper part of \( y \), then the opposite does not hold
Disjoint properties

• E.g: you can’t be both the *parent of* and *child of* the same person

• DisjointObjectProperties

• DisjointDataProperties
  – E.g., startTime and endTime of a surgery
Property chain inclusion

• Properties can be defined as a composition of other properties
• The brother of your parent is your uncle
  :uncle owl:propertyChainAxiom (:parent :brother)
• Don’t be confused: before we defined these things a concepts, but now as roles!
Keys

• Individuals can be identified uniquely
• Identification can be done using
  – A data property
  – An object property or
  – A set of properties
• Example
  
  foaf:Person owl:hasKey (foaf:mbox);
  owl:hasKey (:homePhone :foaf:name).
Extended datatypes

• Extra datatypes
  – Examples: owl:real, owl:rational, xsd:pattern

• Datatype restrictions
  – Range of datatypes
  – For example, adult has an age >= 18
    – DatatypeRestriction(xsd:integer minInclusive 18)

• Datatype definitions
  – New datatypes
    – DatatypeDefinition( :adultAge DatatypeRestriction(xsd:integer minInclusive 18))
Extended datatypes

• Data range combinations
  – Intersection of
    • DataIntersectionOf( xsd:nonNegativeInteger xsd:nonPositiveInteger )
  – Union of
    • DataUnionOf( xsd:string xsd:integer )
  – Complement of data range
    • DataComplementOf( xsd:positiveInteger )
An example

:Teenager rdfs:subClassOf _:x.

_:x rdf:type owl:Restriction ;
    owl:onProperty :hasAge ;
    owl:someValuesFrom _:y.

_:y rdf:type rdfs:Datatype ;
    owl:onDatatype xsd:integer ;
    owl:withRestrictions ( _:z1 _:z2 ) .

_:z1 xsd:minInclusive "13"^^xsd:integer .

_:z2 xsd:maxInclusive "19"^^xsd:integer .
Punning

• An *OWL 1 DL* thing can’t be both a class and an instance
  – E.g., :SnowLeopard can’t be both a subclass of :Feline and an instance of :EndangeredSpecies

• *OWL 2 DL* offers better support for meta-modeling via *punning*
  – A URI denoting an owl thing can have two distinct views, e.g., as a class and as an instance
  – The one intended is determined by its use
  – A *pun* is often defined as a joke that exploits the fact that a word has two different senses or meanings
Punning Restrictions

• Classes and object properties also can have the same name
  – For example, :mother can be both a property and a class of people

• But classes and datatype properties can not have the same name (why?)

• Also datatype properties and object properties can not have the same name
Punning Example

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

foaf:Person a owl:Class.
:Woman a owl:Class.
:Parent a owl:Class.

:mother a owl:ObjectProperty;
   rdfs:domain foaf:Person;
   rdfs:range foaf:Person .

:mother a owl:Class;
   owl:intersectionOf (:Woman :Parent).

validate via http://owl.cs.manchester.ac.uk/validator/
Annotations

• In OWL annotations comprise information that carries no official meaning
• Some properties in OWL 1 are considered as annotation properties, e.g., owl:comment, rdf:label and rdf:seeAlso
• OWL 1 allowed RDF reification as a way to say things about triples, again w/o official meaning

[a rdf:Statement;
  rdf:subject :Barack_Obama;
  rdf:predicate dbpo:born_in;
  rdf:object :Kenya;
  :certainty “0.01” ].
Annotations

• **OWL 2** has native support for annotations, including
  – Annotations on owl axioms (i.e., triples)
  – Annotations on entities (e.g., a Class)
  – Annotations on annotations

• The mechanism is again reification
Annotations

_:Man rdfs:subClassOf :Person .
_:x rdf:type owl:Axiom ;
   owl:subject :Man ;
   owl:predicate rdfs:subClassOf ;
   owl:object :Person ;
   :probability "0.99"^^xsd:integer;
   rdfs:label "Every man is a person." .
Inverse object properties

– some object property can be inverse of another property
– For example, partOf and hasPart
– ObjectInverseOf( :partOf ): this expression represents the inverse property of :part of
– This makes writing ontologies easier by avoiding the need to name an inverse
Tableaux algorithm for SROIQ(D)
To sum up ..

• An even more expressive language than OWL 1
• Improved scalability is not addressed at all so far
• So ... ?
OWL Sub-languages

• OWL 1 had sub-languages: OWL FULL, OWL DL and OWL Lite
• OWL FULL is undecidable
• OWL DL is worst case highly intractable
• Even OWL Lite turned out to be not very tractable (EXPTIME-complete)
• OWL 2 introduced three sub-languages, called Profiles, designed for different use cases
OWL Profiles

• Profiles considered
  – Useful computational properties, e.g., reasoning complexity
  – Implementation possibilities, e.g., using RDBs

• There are three profiles
OWL 2 Profiles

OWL 2 defines three different tractable profiles:

- **EL**: polynomial time reasoning for schema and data
  - Useful for ontologies with large conceptual part
- **QL**: fast (logspace) query answering using RDBMs via SQL
  - Useful for large datasets already stored in RDBs
- **RL**: fast (polynomial) query answering using rule-extended DBs
  - Useful for large datasets stored as RDF triples
OWL 2 EL

• A (near maximal) fragment of OWL 2 such that
  – Satisfiability checking is in PTime (**PTime-Complete**)
  – Data complexity of query answering is PTime-Complete

• Based on **EL** family of description logics
  – Existential (someValuesFrom) + conjunction

• It does not allow disjunction and *universal restrictions*

• *Saturation* is an efficient reasoning technique

• It can capture the expressive power used by many large-scale ontologies, e.g., SNOMED CT
Basic Saturation-based Technique

- Normalise ontology axioms to standard form:
  \[ A \subseteq B \quad A \cap B \subseteq C \quad A \subseteq \exists R.B \quad \exists R.B \subseteq C \]

- Saturate using inference rules:
  \[
  \begin{array}{c}
  A \subseteq B \\
  B \subseteq C
  \end{array} \quad \Rightarrow \quad A \subseteq C
  \]
  \[
  \begin{array}{c}
  A \subseteq B \\
  A \subseteq C
  \end{array} \quad \Rightarrow \quad A \subseteq D
  \]
  \[
  \begin{array}{c}
  A \subseteq \exists R.B \\
  B \subseteq C
  \end{array} \quad \Rightarrow \quad \exists R.C \subseteq D
  \]

- Extension to Horn fragment requires (many) more rules
Saturation-based Technique (basics)

Example: infer that a heart transplant is a kind of organ transplant

\[
\text{OrganTransplant} \equiv \text{Transplant} \land \exists \text{site. Organ}
\]

\[
\text{HeartTransplant} \equiv \text{Transplant} \land \exists \text{site. Heart}
\]

\[
\text{Heart} \sqsubseteq \text{Organ}
\]
Saturation-based Technique (basics)

Example:

OrganTransplant $\equiv$ Transplant $\sqcap \exists$site.Organ
HeartTransplant $\equiv$ Transplant $\sqcap \exists$site.Heart

Heart $\sqsubseteq$ Organ
Saturation-based Technique (basics)

Example:

\[
\text{OrganTransplant} \equiv \text{Transplant} \cap \exists \text{site.Organ} \\
\text{HeartTransplant} \equiv \text{Transplant} \cap \exists \text{site.Heart} \\
\text{Heart} \sqsubseteq \text{Organ}
\]

\[
\text{OrganTransplant} \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} \sqsubseteq \exists \text{site.Organ}
\]
Saturation-based Technique (basics)

Example:

\[
\text{OrganTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} \sqsubseteq \text{Organ}
\]

\[
\text{OrganTransplant} \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} \sqsubseteq \exists \text{site.Organ} \\
\exists \text{site.Organ} \sqsubseteq \text{SO} \\
\text{Transplant} \sqcap \text{SO} \sqsubseteq \text{OrganTransplant}
\]
Saturation-based Technique (basics)

Example:

\[
\text{OrganTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} \sqsubseteq \text{Organ}
\]

\[
\text{OrganTransplant} \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} \sqsubseteq \exists \text{site.Organ} \\
\exists \text{site.Organ} \sqsubseteq \text{SO} \\
\text{Transplant} \sqcap \text{SO} \sqsubseteq \text{OrganTransplant}
\]
Saturation-based Technique (basics)

Example:

\[
\begin{align*}
\text{OrganTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} & \sqsubseteq \text{Organ}
\end{align*}
\]

\[
\begin{align*}
\text{OrganTransplant} & \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} & \sqsubseteq \exists \text{site.Organ} \\
\exists \text{site.Organ} & \sqsubseteq \text{SO} \\
\text{Transplant} \sqcap \text{SO} & \sqsubseteq \text{OrganTransplant} \\
\text{HeartTransplant} & \sqsubseteq \text{Transplant} \\
\text{HeartTransplant} & \sqsubseteq \exists \text{site.Heart} \\
\exists \text{site.Heart} & \sqsubseteq \text{SH} \\
\text{Transplant} \sqcap \text{SH} & \sqsubseteq \text{HeartTransplant}
\end{align*}
\]
Saturation-based Technique (basics)

Example:

\[
\text{OrganTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} \sqsubseteq \text{Organ}
\]

\[
\text{OrganTransplant} \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} \sqsubseteq \exists \text{site.Organ} \\
\exists \text{site.Organ} \sqsubseteq \text{SO} \\
\text{Transplant} \sqcap \text{SO} \sqsubseteq \text{OrganTransplant} \\
\text{HeartTransplant} \sqsubseteq \text{Transplant} \\
\text{HeartTransplant} \sqsubseteq \exists \text{site.Heart} \\
\exists \text{site.Heart} \sqsubseteq \text{SH} \\
\text{Transplant} \sqcap \text{SH} \sqsubseteq \text{HeartTransplant}
\]
Saturation-based Technique (basics)

Example:

\[
\begin{align*}
\text{OrganTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} & \sqsubseteq \text{Organ}
\end{align*}
\]

\[
\begin{align*}
\text{OrganTransplant} & \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} & \sqsubseteq \exists \text{site.Organ} \\
\exists \text{site.Organ} & \sqsubseteq \text{SO} \\
\text{Transplant} \sqcap \text{SO} & \sqsubseteq \text{OrganTransplant} \\
\text{HeartTransplant} & \sqsubseteq \text{Transplant} \\
\text{HeartTransplant} & \sqsubseteq \exists \text{site.Heart} \\
\exists \text{site.Heart} & \sqsubseteq \text{SH} \\
\text{Transplant} \sqcap \text{SH} & \sqsubseteq \text{HeartTransplant} \\
\text{Heart} & \sqsubseteq \text{Organ}
\end{align*}
\]
Saturation-based Technique (basics)

Example:

\[\begin{align*}
\text{OrganTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \\
\text{HeartTransplant} & \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \\
\text{Heart} & \sqsubseteq \text{Organ}
\end{align*}\]

\[\begin{align*}
\text{OrganTransplant} & \sqsubseteq \text{Transplant} \\
\text{OrganTransplant} & \sqsubseteq \exists \text{site.Organ} \\
\exists \text{site.Organ} & \sqsubseteq \text{SO} \\
\text{Transplant} \sqcap \text{SO} & \sqsubseteq \text{OrganTransplant} \\
\text{HeartTransplant} & \sqsubseteq \text{Transplant} \\
\text{HeartTransplant} & \sqsubseteq \exists \text{site.Heart} \\
\exists \text{site.Heart} & \sqsubseteq \text{SH} \\
\text{Transplant} \sqcap \text{SH} & \sqsubseteq \text{HeartTransplant} \\
\text{Heart} & \sqsubseteq \text{Organ}
\end{align*}\]
Saturation-based Technique (basics)

Example:

OrganTransplant ≡ Transplant ∩ ∃site.Organ
HeartTransplant ≡ Transplant ∩ ∃site.Heart
Heart ⊆ Organ

OrganTransplant ⊆ Transplant
OrganTransplant ⊆ ∃site.Organ
∃site.Organ ⊆ SO
Transplant ∩ SO ⊆ OrganTransplant
HeartTransplant ⊆ Transplant
HeartTransplant ⊆ ∃site.Heart
∃site.Heart ⊆ SH
Transplant ∩ SH ⊆ HeartTransplant
Heart ⊆ Organ

\[ A ⊆ ∃R.B \quad B ⊆ C \quad ∃R.C ⊆ D \]
\[ A ⊆ D \]
Saturation-based Technique (basics)

Example:

OrganTransplant $\equiv$ Transplant $\sqcap \exists \text{site.Organ}$

HeartTransplant $\equiv$ Transplant $\sqcap \exists \text{site.Heart}$

Heart $\sqsubseteq$ Organ

$A \subseteq B$ $A \subseteq C$ $B \cap C \subseteq D$

$A \subseteq D$

OrganTransplant $\sqsubseteq$ Transplant

OrganTransplant $\sqsubseteq \exists \text{site.Organ}$

$\exists \text{site.Organ} \sqsubseteq \text{SO}$

Transplant $\sqcap \text{SO} \sqsubseteq \text{OrganTransplant}$

HeartTransplant $\sqsubseteq$ Transplant

HeartTransplant $\sqsubseteq \exists \text{site.Heart}$

$\exists \text{site.Heart} \sqsubseteq \text{SH}$

Transplant $\sqcap \text{SH} \sqsubseteq \text{HeartTransplant}$

Heart $\sqsubseteq$ Organ
Saturation-based Technique (basics)

Example:

\[ \text{OrganTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Organ} \]
\[ \text{HeartTransplant} \equiv \text{Transplant} \sqcap \exists \text{site.Heart} \]
\[ \text{Heart} \sqsubseteq \text{Organ} \]

\[ A \sqsubseteq B \quad A \sqsubseteq C \quad B \sqcap C \sqsubseteq D \quad A \sqsubseteq D \]

\[ \text{OrganTransplant} \sqsubseteq \text{Transplant} \]
\[ \exists \text{site.Organ} \sqsubseteq \text{SO} \]
\[ \text{HeartTransplant} \sqsubseteq \exists \text{site.Organ} \sqsubseteq \text{SO} \]
\[ \text{Transplant} \sqcap \text{SO} \sqsubseteq \text{OrganTransplant} \]
\[ \text{HeartTransplant} \sqsubseteq \text{Transplant} \]
\[ \exists \text{site.Heart} \sqsubseteq \text{SH} \]
\[ \text{Transplant} \sqcap \text{SH} \sqsubseteq \text{HeartTransplant} \]
\[ \text{Heart} \sqsubseteq \text{Organ} \]
# Saturation-based Technique

Performance with large bio-medical ontologies

<table>
<thead>
<tr>
<th>Concepts:</th>
<th>GO</th>
<th>NCI</th>
<th>Galen v.0</th>
<th>Galen v.7</th>
<th>SNOMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACT++</td>
<td>15.24</td>
<td>6.05</td>
<td>465.35</td>
<td>—</td>
<td>650.37</td>
</tr>
<tr>
<td>HERMIT</td>
<td>199.52</td>
<td>169.47</td>
<td>45.72</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PELLET</td>
<td>72.02</td>
<td>26.47</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>CEL</td>
<td>1.84</td>
<td>5.76</td>
<td>—</td>
<td>—</td>
<td>1185.70</td>
</tr>
</tbody>
</table>
OWL 2 QL

• The QL acronym reflects its relation to the standard relational Query Language
• It does not allow existential and universal restrictions to a class expression or a data range
• These restrictions
  – enable a tight integration with RDBMSs,
  – reasoners can be implemented on top of standard relational databases
• Can answer complex queries (in particular, unions of conjunctive queries) over the instance level (ABox) of the DL knowledge base
OWL 2 QL

We can exploit **query rewriting** based reasoning technique

– Computationally optimal
– Data storage and query evaluation can be delegated to standard RDBMS
– Can be extended to more expressive languages (beyond AC$^0$) by delegating query answering to a Datalog engine
Query Rewriting Technique (basics)

• Given ontology $O$ and query $Q$, use $O$ to rewrite $Q$ as $Q^0$ such that, for any set of ground facts $A$:
  
  \[ \text{ans}(Q, O, A) = \text{ans}(Q^0, ;, A) \]

• Resolution based query rewriting
  
  – **Clausify** ontology axioms
  
  – **Saturate** (clausified) ontology and query using resolution
  
  – **Prune** redundant query clauses
Query Rewriting Technique (basics)

• Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
\]
Query Rewriting Technique (basics)

• Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[
treats(x, f(x)) \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) \leftarrow \text{Consultant}(x)
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
\]
Query Rewriting Technique (basics)

• Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor}
\end{align*}
\]

\[
\begin{align*}
treats(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
\end{align*}
\]
Query Rewriting Technique (basics)

- Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))
\end{align*}
\]
Query Rewriting Technique (basics)

• Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats. Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[
treats(x, f(x)) \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) \leftarrow \text{Consultant}(x)
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))
\]
Query Rewriting Technique (basics)

• Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x)
\end{align*}
\]
Query Rewriting Technique (basics)

• Example:

\[ \text{Doctor} \sqsubseteq \exists \text{treats.Patient} \]
\[ \text{Consultant} \sqsubseteq \text{Doctor} \]

\[ \text{treats}(x, f(x)) \leftarrow \text{Doctor}(x) \]
\[ \text{Patient}(f(x)) \leftarrow \text{Doctor}(x) \]
\[ \text{Doctor}(x) \leftarrow \text{Consultant}(x) \]

\[ Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \]
\[ Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \]
\[ Q(x) \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \]
Query Rewriting Technique (basics)

• Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
\text{Q}(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
\text{Q}(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
\text{Q}(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
\text{Q}(x) & \leftarrow \text{Doctor}(x)
\end{align*}
\]
Query Rewriting Technique (basics)

• Example:

\[
\text{Doctor} \subseteq \exists \text{treats.Patient}
\]
\[
\text{Consultant} \subseteq \text{Doctor}
\]

\[
\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)
\]
\[
\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)
\]
\[
\text{Doctor}(x) \leftarrow \text{Consultant}(x)
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
\]
\[
Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))
\]
\[
Q(x) \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x)
\]
\[
Q(x) \leftarrow \text{Doctor}(x)
\]
Query Rewriting Technique (basics)

• Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor}
\end{align*}
\]

\[
\begin{align*}
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]
Query Rewriting Technique (basics)

• Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats.Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor}
\end{align*}
\]

\[
\begin{align*}
treats(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]
Query Rewriting Technique (basics)

• Example:

$$\text{Doctor} \sqsubseteq \exists \text{treats.Patient}$$
$$\text{Consultant} \sqsubseteq \text{Doctor}$$

$$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$$
$$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$$
$$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$$

$$Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)$$
$$Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))$$
$$Q(x) \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x)$$
$$Q(x) \leftarrow \text{Doctor}(x)$$
$$Q(x) \leftarrow \text{Consultant}(x)$$
Query Rewriting Technique (basics)

• Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats}.\text{Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Consultant}(x)
\end{align*}
\]

• For DL-Lite, result is a union of conjunctive queries (UCQ)
Query Rewriting Technique (basics)

• Data can be stored/left in **RDBMS**

• Relationship between ontology and DB defined by **mappings**, e.g.:

  Doctor  $\mapsto$  SELECT Name FROM Doctor
  Patient  $\mapsto$  SELECT Name FROM Patient
  treats  $\mapsto$  SELECT DName, PName FROM Treats

• UCQ translated into **SQL query**:

  SELECT Name FROM Doctor UNION
  SELECT DName FROM Treats, Patient WHERE PName=Name
OWL 2 RL

• The RL acronym reflects its relation to *Rule Languages*.

• Not allowed: existential quantification to a class, union and disjoint union to class expressions.

• These restrictions allow OWL 2 RL to be implemented using rule-based technologies such as rule extended DBMSs, Jess, Prolog, etc.
OWL 2 RL - Example

• $\exists\text{parentOf.} \exists\text{parentOf.} \sqsubseteq \text{Grandfather}$
  
  rule version: $\text{parentOf}(x,y) \land \text{parentOf}(y,z) \rightarrow \text{Grandfather}(x)$

• $\text{Orphan} \sqsubseteq \forall\text{hasParent.} \text{Dead}$
  
  rule version: $\text{Orphan}(x) \land \text{hasParent}(x,y) \rightarrow \text{Dead}(y)$

• $\text{Monogamous} \sqsubseteq \leq \text{I married.} \text{Alive}$
  
  rule version:
  
  $\text{Monogamous}(x) \land \text{married}(x,y) \land \text{Alive}(y) \land \text{married}(x,z) \land \text{Alive}(z) \rightarrow y=z$

• $\text{childOf} \circ \text{childOf} \sqsubseteq \text{grandchildOf}$
  
  rule version: $\text{childOf}(x,y) \land \text{childOf}(y,z) \rightarrow \text{grandchildOf}(x,z)$

• $\text{Disj(childOf, parentOf)}$
  
  rule version: $\text{childOf}(x,y) \land \text{parentOf}(x,y) \rightarrow$
Profiles

Profile selection depends on

– Expressivity required by the application
– Priority given to reasoning on classes or data
– Size of the datasets
OWL 2 Web Ontology Language Quick Reference Guide
http://www.w3.org/2007/OWL/refcard

1 Names, Prefixes, and Notation
Names in OWL 2 are IRIs, often written in a shorthand prefixed_name, where prefix is a prefix namespace that expands to an IRI, and local_name is the remainder of the name. The prefix names in OWL 2 are:

<table>
<thead>
<tr>
<th>Prefix Name</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
</tr>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
</tr>
<tr>
<td>owl</td>
<td><a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a></td>
</tr>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
</tr>
</tbody>
</table>

We use notation conventions in the following table:

<table>
<thead>
<tr>
<th>Letters</th>
<th>Meaning</th>
<th>Letters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>i</em></td>
<td><em>i</em></td>
<td><em>n</em></td>
<td>non-negative integer**</td>
</tr>
<tr>
<td><em>a</em></td>
<td>anonymous individual</td>
<td><em>P</em></td>
<td>property expression</td>
</tr>
<tr>
<td><em>x</em></td>
<td>blank node</td>
<td><em>o</em></td>
<td>object property expression</td>
</tr>
<tr>
<td>A</td>
<td>annotation property</td>
<td><em>R</em></td>
<td>data property</td>
</tr>
<tr>
<td>C</td>
<td>cardinality</td>
<td><em>s</em></td>
<td>OWL interchange</td>
</tr>
<tr>
<td>CN</td>
<td>class name</td>
<td><em>t</em></td>
<td>OWL interchange</td>
</tr>
<tr>
<td>D</td>
<td>data range</td>
<td><em>v</em></td>
<td>literal</td>
</tr>
<tr>
<td>DN</td>
<td>datatype name</td>
<td><em>i</em></td>
<td>non-negative integer**</td>
</tr>
<tr>
<td>f</td>
<td>function</td>
<td><em>n</em></td>
<td>non-negative integer**</td>
</tr>
</tbody>
</table>

* All of the above can have subscripts. ** As a shorthand for “*n* *n* *n* non-negative integer

2 OWL 2 Constructs and Axioms
In the following table, the three columns are:

<table>
<thead>
<tr>
<th>Language Feature</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>For an OWL 2 DL ontology, there are additional global restrictions on axioms.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.1 Class Expressions

<table>
<thead>
<tr>
<th>Predetermined and Named Classes</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>named class CN</td>
<td>CN</td>
<td>CN</td>
</tr>
<tr>
<td>universal class owl:Thing</td>
<td>owl:Thing</td>
<td>owl:Thing</td>
</tr>
<tr>
<td>empty class owl:Nothing</td>
<td>owl:Nothing</td>
<td>owl:Nothing</td>
</tr>
</tbody>
</table>

Boolean Connectives and Enumeration of Individuals

<table>
<thead>
<tr>
<th>Intersection</th>
<th>x rdfs:type owl:Class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>complement</td>
<td>x rdfs:type owl:Class.</td>
</tr>
<tr>
<td>enumeration</td>
<td>x rdfs:type owl:Class.</td>
</tr>
</tbody>
</table>

Object Property Restrictions

<table>
<thead>
<tr>
<th>universal</th>
<th>x rdfs:subClassOf</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>existential</td>
<td>x rdfs:subClassOf</td>
<td></td>
</tr>
</tbody>
</table>

2.2 Properties

<table>
<thead>
<tr>
<th>Object Property Expressions</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>named object property</td>
<td>owl:topObjectProperty</td>
<td>owl:topObjectProperty</td>
</tr>
<tr>
<td>universal object property</td>
<td>owl:topObjectProperty</td>
<td>owl:topObjectProperty</td>
</tr>
</tbody>
</table>

Data Property Expressions

<table>
<thead>
<tr>
<th>named data property</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal data property</td>
<td>owl:topDataProperty</td>
<td>owl:topDataProperty</td>
</tr>
</tbody>
</table>

2.3 Individuals & Literals

<table>
<thead>
<tr>
<th>individual</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>anonymous individual</td>
<td><em>i</em></td>
<td><em>i</em></td>
</tr>
</tbody>
</table>

2.4 Data Ranges

<table>
<thead>
<tr>
<th>Data Range Expressions</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal</td>
<td>x rdfs:subClassOf</td>
<td>owl:Datatype.</td>
</tr>
<tr>
<td>existential</td>
<td>x rdfs:subClassOf</td>
<td>owl:Datatype.</td>
</tr>
</tbody>
</table>

2.5 Axioms

<table>
<thead>
<tr>
<th>Class Expression Axioms</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>subclass</td>
<td>C1 rdfs:subClassOf</td>
<td>C2</td>
</tr>
<tr>
<td>equivalent classes</td>
<td>C1 rdfs:equivalentClass</td>
<td>C2</td>
</tr>
<tr>
<td>disjoint classes</td>
<td>C1 rdfs:disjointWith</td>
<td>C2</td>
</tr>
<tr>
<td>partOf</td>
<td>C1 rdfs:partOfClass</td>
<td>C2</td>
</tr>
<tr>
<td>disjoint union</td>
<td>C1 rdfs:disjointUnion</td>
<td>C2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object Property Axioms</th>
<th>Functional Syntax</th>
<th>RDF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>subproperty</td>
<td>C1 rdfs:subProperty</td>
<td>P2</td>
</tr>
</tbody>
</table>

2.6 Data Property Axioms

| x rdfs:subPropertyOf | P2 |

2.7 Restrictions Using n-ary Data Range

<table>
<thead>
<tr>
<th>n-ary universal</th>
<th>x rdfs:subPropertyOf</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>existential</td>
<td>x rdfs:subPropertyOf</td>
<td>P2</td>
</tr>
</tbody>
</table>

2.8 Equivalent Classes

| x rdfs:equivalentClass | C1 |

2.9 Disjoint Classes

| x rdfs:disjointWith | C1 |

2.10 Disjoint Union

| x rdfs:disjointUnion | C1 |

2.11 Restrictions Using n-ary Data Range

<table>
<thead>
<tr>
<th>x rdfs:subPropertyOf</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>existential</td>
<td>x rdfs:subPropertyOf</td>
</tr>
<tr>
<td>inverse properties</td>
<td>P1 x rdfs:inverseProperty</td>
</tr>
</tbody>
</table>

In the following table *n1* is an n-ary range type.
# Key OWL 2 Documents

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>For Users</td>
<td><strong>Document Overview.</strong> A quick overview of the OWL 2 specification that includes a description of its relationship to OWL 1. This it the starting point and primary reference point for OWL 2.</td>
</tr>
<tr>
<td>2</td>
<td>Core Spec</td>
<td><strong>Structural Specification and Functional-Style Syntax</strong> defines the constructs of OWL 2 ontologies in terms of both their structure and a functional-style syntax, and defines OWL 2 DL ontologies in terms of global restrictions on OWL 2 ontologies.</td>
</tr>
<tr>
<td>3</td>
<td>Core Spec</td>
<td><strong>Mapping to RDF Graphs</strong> defines a mapping of the OWL 2 constructs into RDF graphs, and thus defines the primary means of exchanging OWL 2 ontologies in the Semantic Web.</td>
</tr>
<tr>
<td>4</td>
<td>Core Spec</td>
<td><strong>Direct Semantics</strong> defines the meaning of OWL 2 ontologies in terms of a model-theoretic semantics.</td>
</tr>
<tr>
<td>5</td>
<td>Core Spec</td>
<td><strong>RDF-Based Semantics</strong> defines the meaning of OWL 2 ontologies via an extension of the RDF Semantics.</td>
</tr>
<tr>
<td>6</td>
<td>Core Spec</td>
<td><strong>Conformance</strong> provides requirements for OWL 2 tools and a set of test cases to help determine conformance.</td>
</tr>
<tr>
<td>7</td>
<td>Specification</td>
<td><strong>Profiles</strong> defines three sub-languages of OWL 2 that offer important advantages in particular applications scenarios.</td>
</tr>
<tr>
<td>8</td>
<td>For Users</td>
<td><strong>OWL 2 Primer</strong> provides an approachable introduction to OWL 2, including orientation for those coming from other disciplines.</td>
</tr>
<tr>
<td>9</td>
<td>For Users</td>
<td><strong>OWL 2 New Features and Rationale</strong> provides an overview of the main new features of OWL 2 and motivates their inclusion in the language.</td>
</tr>
<tr>
<td>10</td>
<td>For Users</td>
<td><strong>OWL 2 Quick Reference Guide</strong> provides a brief guide to the constructs of OWL 2, noting the changes from OWL 1.</td>
</tr>
<tr>
<td>11</td>
<td>Specification</td>
<td><strong>XML Serialization</strong> defines an XML syntax for exchanging OWL 2 ontologies, suitable for use with XML tools like schema-based editors and XQuery/XPath.</td>
</tr>
<tr>
<td>12</td>
<td>Specification</td>
<td><strong>Manchester Syntax</strong> (WG Note) defines an easy-to-read, but less formal, syntax for OWL 2 that is used in some OWL 2 user interface tools and is also used in the Primer.</td>
</tr>
<tr>
<td>13</td>
<td>Specification</td>
<td><strong>Data Range Extension: Linear Equations</strong> (WG Note) specifies an optional extension to OWL 2 which supports advanced constraints on the values of properties.</td>
</tr>
</tbody>
</table>

Conclusion

• Most of the new features of OWL 2 in comparing with the initial version of OWL have been discussed

• Rationale behind the inclusion of the new features have also been discussed

• Three profiles – OWL 2 EL, OWL 2 QL and OWL 2 RL, and their necessity have been presented
Personal conclusion

• The language-fragment landscape becomes too complicated

• Choice of fragment is a design choice again
  – No easy change at a later stage

• Probably one fragment will turn out to be most valuable
  – Rule Languages (especially Datalog +/-) are candidates with a lot of potential!