SQL, DLs, Datalog, and ASP: comparison

Riccardo Rosati
Knowledge Representation and Semantic Technologies
Corso di Laurea in Ingegneria informatica
Sapienza Università di Roma
2016/2017
CWA vs. OWA

• DLs are based on the semantics of classical logic
• Open-world assumption (OWA):
  – The knowledge expressed by the theory is not complete
  – Many possible worlds (models)
• Databases are based on a closed-world assumption (CWA):
  – The knowledge expressed by the database is complete
  – Only one world is possible (the one completely described by the database)
• Datalog and ASP inherit CWA from databases
• But: ASP is able to deal with incomplete knowledge (disjunction)
Complete vs. incomplete knowledge

• DL ABoxes are **incomplete specifications** of the domain of interest
• TBox can be used to infer implicit facts about the world
  – E.g., from
    
    \[ \text{ABox} = \{ \text{student(Bob)} \} \text{ and} \]
    
    \[ \text{TBox} = \{ \text{subClassOf(student, person)} \} \]

  we conclude \text{person(Bob)} (which is NOT stated in the ABox)
  
  – The knowledge expressed by the database is complete
  – Only one world is possible (the one completely described by the database)
Complete vs. incomplete knowledge

• In relational DBs, the database instance is a complete specification:
  – Schema constraints in databases are **integrity constraints**: they MUST be satisfied by the database instance (otherwise the database is in an illegal state)
  – For this reason, such constraints cannot derive new facts (not appearing in the database instance)
Recursion

• DLs are able to deal with recursive statements
  – E.g., cyclic definitions of concepts in the TBox
• DLs do not allow for recursive queries
• SQL does not allow for recursive queries
  – only a limited form of recursion in views is allowed
• Datalog allows for expressing recursive queries over databases
• ASP allows for expressing recursive queries over databases
Negation

• Several DLs allow for expressing negation in the TBox (and in the ABox) and in queries
• But, DLs do not allow for recursive negation in queries
• SQL allows for negation in queries, but does not allow for recursive negation
• Datalog does not allow for expressing negation
• ASP allows for expressing negation, and even recursive negation
Value invention

- DLs allow for expressing so-called value invention (or mandatory participation to roles)
- Example: inclusion axiom $\text{Person} \subseteq \exists \text{hasMother} \cdot T$
- The above axiom cannot be represented in Datalog
- E.g., the rule
  
  $\text{hasMother}(x,y) :- \text{Person}(x)$.

  is not range-restricted (so it is not allowed in Datalog)

- Datalog does not allow for talking about elements outside the Herbrand Universe of the program (namely, the constants appearing in the program)
- Same restriction holds for ASP (and SQL too)
- Value invention is a form of incomplete knowledge
Existential rules

• To overcome the above limitation, an extension of Datalog called *existential rules*, or *Datalog +/-*, has been proposed.

• Existential rules do not have the range restriction on variables: a variable can appear in the head of the rule.

• Such variables are interpreted as existentially quantified variables.

• E.g.:

```
hasMother(x,y) :- Person(x).
```

is an existential rule, whose meaning is: if x is a person, then there exists y such that x has mother y.

• In this way, value invention can be expressed by rules.
Existential rules

• Extending Datalog with existential variables in the head makes the language more expressive
• On the other hand, such an extension makes reasoning harder: in general, reasoning with existential rules is **undecidable**
• Recent studies have defined restricted classes of existential rules in which reasoning is decidable (and even tractable in data complexity)
• Interesting relationship between some of these classes and Description Logics
Comparison: expressiveness

<table>
<thead>
<tr>
<th></th>
<th>Semantics</th>
<th>Recursion</th>
<th>NOT (stratif.)</th>
<th>NOT (recursive)</th>
<th>OR</th>
<th>Value invention</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL</strong></td>
<td>CWA</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><strong>DL</strong></td>
<td>OWA</td>
<td>yes/no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>Datalog</strong></td>
<td>CWA</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Datalog + stratif. negation</strong></td>
<td>CWA</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>ASP</strong></td>
<td>CWA</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
Complexity of SQL

- Complexity of evaluating queries in SQL is NP-complete
- A query may have an exponential number of answers
- Example:

  **table T:**
  
<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

  **query:**
  
  from T AS R1, T AS R2, T AS R3, ..., T AS Rn

  **answers:**
  (2^n tuples)

<table>
<thead>
<tr>
<th>R1.A</th>
<th>R2.A</th>
<th>R3.A</th>
<th>...</th>
<th>Rn-1.A</th>
<th>Rn.A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Complexity of SQL (cont.)

Previous query SQL:
from T AS R1, T AS R2, T AS R3, ..., T AS Rn

Same query expressed in Datalog:
Q(X1, X2, ...,Xn):- T(x1), T(x2), ..., T(Xn).

(which shows that Datalog can build IDB predicates with an exponential number of tuples in the minimal model)
## Comparison: complexity

<table>
<thead>
<tr>
<th>Reasoning task</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL</strong></td>
<td>Query evaluation</td>
</tr>
<tr>
<td>ALC, unfoldable TBoxes</td>
<td>Concept consistency, KB satisfiability, instance checking</td>
</tr>
<tr>
<td>ALC, GCIs</td>
<td>Concept consistency, KB satisfiability, instance checking</td>
</tr>
</tbody>
</table>
Comparison: complexity

<table>
<thead>
<tr>
<th></th>
<th>Reasoning task</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground positive Datalog</td>
<td>Building the minimal model</td>
<td>PTIME-complete</td>
</tr>
<tr>
<td>Ground Datalog with stratified negation</td>
<td>Building the minimal model</td>
<td>PTIME-complete</td>
</tr>
<tr>
<td>Ground ASP (no disjunction)</td>
<td>Consistency, brave reasoning</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Ground ASP</td>
<td>Consistency, brave reasoning</td>
<td>NP&lt;sup&gt;NP&lt;/sup&gt;-complete</td>
</tr>
</tbody>
</table>
## Comparison: complexity

<table>
<thead>
<tr>
<th>Model</th>
<th>Reasoning task</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive Datalog</td>
<td>Building the minimal model</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>Datalog with stratified negation</td>
<td>Building the minimal model</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>ASP (no disjunction)</td>
<td>Consistency, brave reasoning</td>
<td>NEXPTIME-complete</td>
</tr>
<tr>
<td>ASP</td>
<td>Consistency, brave reasoning</td>
<td>NEXPTIME^{NP}-complete</td>
</tr>
</tbody>
</table>