Datalog-Based Data Access over Ontology Knowledge Bases

Unit 4 – Systems

Thomas Eiter

Institut für Informationsysteme, TU Wien

ICCL Summer School 2013, August 29-30, 2013
Unit Outline

1. Introduction
2. Clipper
3. DReW – Datalog Rewriting System
4. Conclusion
Some Systems / Tools

- General queries can be reduced to Boolean queries: $Q(\vec{X}) \sim Q(\vec{c})$
- This is not much practical
- Systems handling *non-ground* queries, as in databases, are needed

Predominant: rewriting to relational DBMS

**FO-Rewriting Systems**

- *Quonto / Presto*
- *Owlgres*
- *Requiem/DL-Lite*
- *MOR (dl-progs/DL-Lite)*

**Datalog-Rewriting Systems**

- *Requiem/EL*
- *Clipper*
- *Aspide/OWL-2-DLVex, Aspide/Requiem*
- *DReW (dl-progs)*
**CLIPPER Reasoner**

- **CLIPPER** is a reasoner for CQ answering over Horn-\(SHIQ\) ontologies

- **Homepage**: http://www.kr.tuwien.ac.at/research/systems/clipper
  
  at GitHub: https://github.com/ghxiao/clipper

- Based on Datalog rewriting (unit 3)

- Implemented in Java

- Ontology parser: OWL-API

- Datalog backend: DLV (inside CLIPPER); Clingo may be used as well (compute rewriting, via command line)
Query Answering Algorithm

Recall: three steps to construct Datalog program

**Algorithm** Horn-\(\text{SHIQ}\)-CQ

---

**Input**: normal Horn-\(\text{SHIQ}\) KB \(\mathcal{K} = (\mathcal{T}, \mathcal{A})\), conjunctive query \(Q\)

**Output**: query answers

\[
\Xi(\mathcal{T}) \leftarrow \text{Saturate}(\mathcal{T}); \\
\text{rew}_\mathcal{T}(Q) \leftarrow \text{Rewrite}(Q, \Xi(\mathcal{T})); \\
\text{cr}(\mathcal{T}) \leftarrow \text{CompletionRules}(\mathcal{T}); \\
P \leftarrow \mathcal{A} \cup \text{cr}(\mathcal{T}) \cup \text{rew}_\mathcal{T}(q); \\
\text{ans} \leftarrow \{\vec{u} \mid q(\vec{u}) \in \text{Datalog-eval}(P)\}; \quad \triangleright \text{call Datalog reasoner}
\]

**Note:**

- recursive rules might occur in \(\text{CR}(\mathcal{T})\)
- predicate arities in \(P\) are bounded by 2.
- hence rewriting \(P\) is necessarily exponential in the worst case (unless \(\text{EXPTIME} = \text{NP}\))
System Architecture

- Version 0.1: CQs with simple roles only (no transitive roles)
- Use SPARQL syntax
Usage

Usage: clipper.sh [options] [command] [command options]

Options:
- `v, -verbose` Level of verbosity
  Default: 1

Commands:
query answerting conjunctive query
Usage: query [options] <ontology.owl> -sparql <cq.sparql>

Options:
- `-f, --output-format` output format, possible values: { table | csv | atoms | html }
  Default: table
- `-dlv` the location of dlv (e.g. /usr/local/bin/dlv)

rewrite rewrite the query w.r.t. the ontology, and generate a datalog program
Usage, cont’d

Usage: rewrite [options] <ontology.owl> [ -sparql <cq.sparql> ]

Options:
- --abox-only, -a only rewrite ABox
  Default: false
- --ontology-and-query, -oq rewrite ontology (= TBox + ABox) and query
  Default: false
- --ontology-only, -o only rewrite ontology (= TBox + ABox)
  Default: false
- --remove-redundancy, -r remove redundancy rules w.r.t the query
  Default: false
- --tbox-and-query, -tq only rewrite TBox and query
  Default: false
- --tbox-only, -t only rewrite TBox
  Default: false
- -output-datalog, -d output datalog file
Properties

- Comparison with state-of-the-art systems PRESTO and REQUIEM [E_ et al. 2012a,2012b]
- CLIPPER shows promising results
- Nice downscaling behavior, on less expressive ontologies (falling into $DL$-Lite)
- Prototype with transitive roles may be available soon!
- Possible extensions (projected)
  - weakly DL-safe rules (algorithm developed, implementation pending)
  - other DLs, like regular $EL^{++}$ and Horn-$SRIQ$, datatypes
  - more expressive queries, like regular path queries

Big issue: lack of realistic test cases!
DReW System

Loose Coupling - revisited

- **Advantage:**
  - clean semantics, can use legacy systems
  - fairly easy to incorporate further knowledge formats (e.g. RDF)
  - supportive to privacy, information hiding

- **Drawback:** *impedance mismatch, performance*
  - dl-program evaluation needs multiple calls of a dl-reasoner
  - Calls are expensive
    * optimizations (caching, pruning ...)
  - exponentially many calls may be unavoidable
  - Even polynomially many calls might be too costly
Uniform Evaluation

Method

Convert the evaluation problem into one for a single reasoning engine

- Transform dl-program \( \Pi \) into an (equivalent) knowledge base in formalism \( \mathcal{L} \) for evaluation (uniform evaluation)
  - \( \mathcal{L} = \text{FO Logic (SQL): MOR; acyclic } \Pi \text{ over } DL\text{-Lite}, \text{ using an RDBMS} \)
  - \( \mathcal{L} = \text{Datalog}\neg: \text{DReW; } \Pi \text{ over Datalog-rewritable ontologies} \)

Note: uniform evaluation is different from tight integration in a unifying logic
Issues

“Uniform Evaluation” raises some issues:

1) **Cost of a transformation.** E.g.,
   - Reduction of CQs over DL-Lite ontologies to
     - first-order (FO) Logic [Calvanese et al., 2007]
     - non-recursive Datalog [Gottlob and Schwentick, 2011], [Kontchakov et al., 2010]
   - Reduction of $SHIQ$ to $DATALOG^\vee$ [Hustadt et al., 2007].

2) **Existence of a transformation** (possibly under constraints)
   - Embedding of a formalism into another
   - Properties (e.g. modularity [Janhunen, 1999])
   - Embedding of $dl$-programs e.g. into MKNF [Motik and Rosati, 2010], Equilibrium Logic [Fink and Pearce, 2010]

3) **Complexity** of the target formalism

4) **Feasibility of transformations** for practical concerns
Uniform Datalog Evaluation of $\mathcal{dl}$-Programs

**Idea:**

- for Datalog-rewritable ontologies, we may replace $\mathcal{dl}$-atoms $DL[\lambda; Q](\vec{c})$ with Datalog programs evaluating the atoms
- the result is computed in an atom $Q_\lambda(\vec{c})$
- rewrite the $\mathcal{dl}$-rules to ordinary rules, by replacing $\mathcal{dl}$-atoms
- evaluate the resulting logic program using a Datalog engine / ASP solver

**Important:** uniform query rewriting, must work for all $\lambda$

Demonstrate the method on the Network example
Network Example

\( \Pi = (\mathcal{O}, P) \)

\begin{align*}
\text{Ontology } & \mathcal{O} : \\
& \geq 1. \text{wired} \sqsubseteq \text{Node} \quad \top \sqsubseteq \forall \text{wired.Node} \\
& \text{wired} = \text{wired}^{-} \\
& n_1 \neq n_2 \neq n_3 \neq n_4 \neq n_5 \\
& \text{wired}(n_1, n_2) \ \text{wired}(n_2, n_3) \ \text{wired}(n_2, n_4) \\
& \text{wired}(n_2, n_5) \ \text{wired}(n_3, n_4) \ \text{wired}(n_3, n_5). \\
& \geq 4. \text{wired} \sqsubseteq \text{HighTrafficNode} \\
\end{align*}

Rules \( P \)

\begin{align*}
\text{newnode}(x_1). \quad \text{newnode}(x_2). \\
\text{overloaded}(X) & \leftarrow \text{DL}[\text{wired} \uplus \text{connect}; \text{HighTrafficNode}](X). \\
\text{connect}(X, Y) & \leftarrow \text{newnode}(X), \text{DL}[\text{Node}](Y), \\
& \quad \text{not overloaded}(Y), \text{not excl}(X, Y). \\
\text{excl}(X, Y) & \leftarrow \text{connect}(X, Z), \text{DL}[\text{Node}](Y), Y \neq Z. \\
\text{excl}(X, Y) & \leftarrow \text{connect}(Z, Y), \text{newnode}(Z), \text{newnode}(X), Z \neq X. \\
\text{excl}(x_1, n_4). 
\end{align*}
Network Example, cont’d

1. Rewriting the ontology

- The DL component $O$ is in OWL 2 RL resp. $\mathcal{LDL}^+$, which is Datalog-rewritable (so are the dl-atoms).
- We transform $O$ to the Datalog program $\Phi_{\mathcal{LDL}^+}(O)$:

\[
\begin{align*}
\text{wired}^- (Y, X) & \leftarrow \text{wired}(X, Y) \\
\top (X) & \leftarrow \text{wired}(X, Y) \\
\top (X) & \leftarrow \text{wired}^- (X, Y)
\end{align*}
\]

%axiom $\geq 1.\text{wired} \sqsubseteq \text{Node}$

\[
\begin{align*}
\text{Node}(Y) & \leftarrow \text{wired}(X, Y)
\end{align*}
\]

%axiom $\top \sqsubseteq \forall \text{wired}.\text{Node}$

\[
\begin{align*}
\text{Node}(Y) & \leftarrow \text{wired}(X, Y), \top (X)
\end{align*}
\]

%axiom $\geq 4.\text{wired} \sqsubseteq \text{HighTrafficNode}$

\[
\begin{align*}
\text{HighTrafficNode}(X) & \leftarrow \text{wired}(X, Y_1), \text{wired}(X, Y_2), \text{wired}(X, Y_3), \text{wired}(X, Y_4), \\
Y_1 & \neq Y_2, Y_1 \neq Y_3, \ldots, Y_3 \neq Y_4.
\end{align*}
\]

\[
\begin{align*}
\text{wired}(n_1, n_2) & \text{ wired}(n_2, n_3) \text{ wired}(n_2, n_4), \text{ wired}(n_2, n_5). \text{ wired}(n_3, n_4). \text{ wired}(n_3, n_5).
\end{align*}
\]
Network Example, cont’d

2. Duplicating for dl-inputs

dl-atoms in Π:

\[ DL[Node](Y), \quad DL[\text{wired} \sqcup \text{connect}; \text{HighTrafficNode}](X) \]

- the dl-queries in are just instance queries, so given by \( Node(Y) \) resp. \( \text{HighTrafficNode}(X) \)

- Each DL-atom sends up a different input \( \lambda \) to \( O \) and so entailments for the \( \lambda \)'s might be different.

- To this purpose, we copy \( \Phi_{LDL^+}(O) \) to new disjoint equivalent versions for each DL-input \( \lambda \)

- For the set \( \Lambda_P = \{ \lambda_1 = \epsilon, \lambda_2 = \text{wired} \sqcup \text{connect} \} \), we have

\[ \Phi_{LDL^+},\lambda_1(O) = \{ \text{Node}_{\lambda_1}(X) \leftarrow \text{wired}_{\lambda_1}(X, Y), \ldots \} \text{ and} \]
\[ \Phi_{LDL^+},\lambda_2(O) = \{ \text{Node}_{\lambda_2}(X) \leftarrow \text{wired}_{\lambda_2}(X, Y), \ldots \} \]
Network Example, cont’d

3. Rewriting dl-rules to ordinary rules

To rewrite DL-rules $P$ into ordinary rules $P^{ord}$, we simply replace each DL-atom $DL[\lambda; Q](\vec{t})$ by a new atom $Q_{\lambda}(\vec{t})$.

\[ P^{ord} \]

\begin{align*}
\text{newnode}(x_1). & \quad \text{newnode}(x_2). \\
\text{overloaded}(X) & \leftarrow \text{HighTrafficNode}_{\lambda_2}(X). \\
\text{connect}(X, Y) & \leftarrow \text{newnode}(X), \text{Node}_{\lambda_1}(Y), \\
& \quad \text{not overloaded}(Y), \text{not excl}(X, Y). \\
\text{excl}(X, Y) & \leftarrow \text{connect}(X, Z), \text{Node}_{\lambda_1}(Y), Y \neq Z. \\
\text{excl}(X, Y) & \leftarrow \text{connect}(Z, Y), \text{newnode}(Z), \text{newnode}(X), Z \neq X. \\
\text{excl}(x_1, n_4). & 
\end{align*}
Network Example, cont’d

4. Rewriting dl-atom Input to Datalog rules

- The inputs $\lambda$ for the copies $\Phi_{\mathcal{DL}L^+,\lambda}$ can be transferred by rules:
  
  - $\lambda_1 = \epsilon$ (no input); no rule needed
  
  - $\lambda_2 = \text{wired} \uplus \text{connect}$:
    
    \[
    \text{wired}_{\lambda_2}(X, Y) \leftarrow \text{connect}(X, Y).
    \]
Network Example, cont’d

5. Calling the Datalog reasoner

- Now we have transformed all the components into a Datalog\neg program

\[ \Psi_{LD\mathcal{L}^+}(\Pi) = \Phi_{LD\mathcal{L}^+},\lambda_1(\Sigma) \cup \Phi_{LD\mathcal{L}^+},\lambda_2(\Sigma) \cup P^{ord} \cup P(\Lambda_P). \]

- We can send it to a datalog engine, e.g. DLV, and compute its answer set or the well-founded model

- The answer sets of \( \Psi_{LD\mathcal{L}^+}(\Pi) \), filtered to *connect*, *overloaded*, *newnode*, *excl*, are the (strong) answer sets of \( \Pi \)

- \( \Psi_{LD\mathcal{L}^+}(\Pi) \models_{wf} p(a) \) iff \( \Pi \models_{wf} p(a) \) for ground atom

Example: \( \Psi_{LD\mathcal{L}^+}(\Pi) \models_{wf} overloaded(n_2) \)
dl-program Transformation (General Case)

\( \mathcal{DL} \): Datalog-rewritable Description Logic

\( \Pi = (\mathcal{O}, P) \): a dl-program with dl-atoms \( DL[\lambda_i; Q_i](\vec{t}_i) \), \( 1 \leq i \leq n \), where

- \( \lambda_i = S_{i,1} \uplus p_{i,1}, \ldots, S_{i,m_i} \uplus p_{i,m_i} \), and
- \( Q_i \) is an instance query.

Let \( \Lambda_P = \{ \lambda_1, \ldots, \lambda_n \} \) and define

\[
\Psi_{\mathcal{DL}}(\Pi) := \bigcup_{\lambda_i \in \Lambda_P} \Phi_{\mathcal{DL},\lambda_i}(\mathcal{O}) \cup P^{ord} \cup \rho(\Lambda_P) \cup T_P
\]

where

- \( \Phi_{\mathcal{DL},\lambda_i}(\mathcal{O}) \) is a copy of \( \Phi_{\mathcal{DL}}(\mathcal{O}) \) with all predicates subscripted with \( \lambda_i \)
- \( \rho(\Lambda_P) \) consists of rules \( S_{i,j,\lambda}(\vec{X}_{i,j}) \leftarrow p_{i,j}(\vec{X}_{i,j}) \), for all \( \lambda_i \in \Lambda_P \)
- \( P^{ord} \) is \( P \) with each \( DL[\lambda_i; Q_i](\vec{t}_i) \) replaced by a new atom \( Q_{\lambda_i}(\vec{t}_i) \)
- \( T_P = \{ \top(a), \top^2(a, b) \mid a, b \text{ occur in } P \} \)
dl-program Transformation (General Case)

Remark: \( \Phi_{DL}(\bigcup_{\lambda_i \in \Lambda_P} O_{\lambda_i}) \) in place of \( \bigcup_{\lambda_i \in \Lambda_P} \Phi_{DL,\lambda_i}(O) \) may be possible, where \( O_{\lambda_i} \) is a copy of \( O \) with all predicates subscripted with \( \lambda_i \).

Theorem

Let \( \Pi = (O, P) \) be a dl-program over Datalog-rewritable \( DL \). Then

1. for every \( a \in HB_P \), \( \Pi |_{wf} a \) iff \( \Psi_{DL}(\Pi) |_{wf} a \);
2. the answer sets of \( \Pi \) correspond 1-1 to the answer sets of \( \Psi(\Pi) \), s.t.
   (i) every strong answer set of \( \Pi \) is expendable to an answer set of \( \Psi(\Pi) \);
   and
   (ii) for every answer set \( J \) of \( \Psi(\Pi) \), its restriction \( I = J |_{HB_p} \) to \( HB_P \) is an answer set of \( \Pi \).

Note:

- updates \( S \cup p \) can be effected via \( \neg S \cup p \) (if permissible, or emulated)
- other queries \( Q \) depending on rewritability
DReW Reasoner

**DReW** prototype: uniform d\(_1\)-program evaluation in Datalog\(^\neg\)

http://www.kr.tuwien.ac.at/research/systems/drew/

at GitHub: https://github.com/ghxiao/drew

- written in Java
- ontology parser: OWL-API
- Datalog reasoner: DLV (inside DReW); Clingo may be used as well (compute rewriting, via command line)

**Features in DReW v0.3**

- ontology component
  - OWL 2 RL (\L DL\(^+\))
  - OWL 2 EL (\S R\(\bigcap\), \times))
- rule formalism
  - d\(_1\)-Programs (answer sets, well founded semantics)
  - CQs under DL-safeness
  - Terminological Default Reasoning (frontend)
System Architecture (Core)
Usage


-rl | -el
  rewriting for OWL 2 RL or OWL 2 EL

-asp, -wf
  the semantics of DL-Programs
  -asp: Answer set semantics (default)
  -wf: Well-founded semantics

<ontology_file>
  the ontology file to be read

<sparql_file>
  the sparql file to be query, which has to be a conjunctive query

<dlp_file>
  the dl-program file

<df_file>
  the default rules file

<dlv_file>
  the path of dlv

<verbose_level>
  Specify verbose category (default: 0)

Example: drew -el -ontology university.owl -dlp rule.dlp -dlv /usr/bin/dlv
Example Usage

Example with Network dl-Program under ASP semantics:

$ ./drew -rl -ontology sample_data/network.owl \
-dlp sample_data/network.dlp \
-filter connect -dlv $HOME/bin/dlv

{ connect(x1, n1)  connect(x2, n5) }

{ connect(x1, n5)  connect(x2, n1) }

{ connect(x1, n5)  connect(x2, n4) }

{ connect(x1, n1)  connect(x2, n4) }
Example Usage, cont’d

Example with network dl-Programs under well-founded semantics

$ ./drew -rl -ontology sample_data/network.owl \ -dlp sample_data/network.dlp \ -filter overloaded -wf -dlv ./dlv-wf

{ overloaded(n2) }
Frontend: Terminological Default Reasoning

- Support for ontologies $L = (\mathcal{T}, \mathcal{A})$ extended with default rules $D$ in the style of terminological default logic [Baader and Hollunder, 1995]

**Example:** access control policy, akin to [Bonatti et al., 2011]

\[
\mathcal{T} = \left\{ \begin{array}{l}
\text{Staff} \sqsubseteq \text{User}, \quad \text{Blacklisted} \sqsubseteq \text{Staff}, \quad \text{Deny} \sqcap \text{Grant} \sqsubseteq \bot, \\
\text{UserRequest} \equiv \exists \text{hasAction}.\text{Action} \sqcap \exists \text{hasSubject}.\text{User} \sqcap \exists \text{hasTarget}.\text{Project}, \\
\text{StaffRequest} \equiv \exists \text{hasAction}.\text{Action} \sqcap \exists \text{hasSubject}.\text{Staff} \sqcap \exists \text{hasTarget}.\text{Project}, \\
\text{BlacklistedStaffRequest} \equiv \text{StaffRequest} \sqcap \exists \text{hasSubject}.\text{Blacklisted}
\end{array} \right\}
\]

\[
\mathcal{A} = \{ \text{StaffRequest}(r1), \quad \text{Blacklisted}(\text{jim}), \ldots \}
\]

\[
D = \left\{ \begin{array}{l}
\text{UserRequest}(X) : \text{Deny}(X)/\text{Deny}(X), \\
\text{StaffRequest}(X) : \neg \text{BlacklistedStaffRequest}(X)/\text{Grant}(X), \\
\text{BlacklistedStaffRequest}(X) : \top/\text{Deny}(X)
\end{array} \right\}
\]

- users normally are denied access to files, staff is granted access
- blacklisted staff are denied any access

- Implements transformation of extended $O^+ = (\mathcal{T}, \mathcal{A}, D)$ to dl-programs (over $\mathcal{EL}$) [Dao-Tran et al., 2009]
Ongoing / Future Work

- More evaluation, use cases
  - Rule-based reasoning over Business ontologies (EDIMine project)
  - Geodata reasoning: semantically enriched spatial queries
- More expressive DL ontology reasoning, e.g. Horn-$SHIQ$
- More reasoning paradigm support, e.g. Closed World Assumption
- Further update operators ($\cap$) and semantics
Conclusion

- Systems using Datalog rewriting are emerging
- DB technology can be fruitfully exploited
- Much to do . . .
- E.g., issue: Datalog\(^\neg\)-rewritability

Relaxed notions of Datalog-rewritability (allow for auxiliary relations)

Datalog\(^\neg\)-rewritability of dl-atoms:

- program \(\Phi_{DL}(\mathcal{O})\) could have multiple answer sets (or none)
- Plugging in \(\Phi_{DL}(\mathcal{O}_{\lambda})\) for some dl-atom \(DL[\lambda, Q](\overrightarrow{t})\) may lead to unwanted effects (e.g., additional answer sets)
  \[\Rightarrow\] use syntactic restrictions (e.g., acyclicity, dl-atoms are not involved in cycles)


Thomas Eiter, Magdalena Ortiz, Mantas Šimkus, Kien Trung-Tran, and Guohui Xiao.  
Query rewriting for Horn-$SHIQ$ plus rules.  

Thomas Eiter, Magdalena Ortiz, Mantas Šimkus, Kien Trung-Tran, and Guohui Xiao.  
Towards practical query answering for Horn-$SHIQ$.  

Michael Fink and David Pearce.  
A logical semantics for description logic programs.  

Georg Gottlob and Thomas Schwentick.  
Rewriting ontological queries into small nonrecursive datalog programs.  
Ullrich Hustadt, Boris Motik, and Ulrike Sattler.  
Reasoning in description logics by a reduction to disjunctive datalog.  

Tomi Janhunen.  
On the intertranslatability of non-monotonic logics.  

Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev.  
The combined approach to query answering in dl-lite.  

B. Motik and R. Rosati.  
Reconciling Description Logics and Rules.  

P. Schneider.  
Evaluation of description logic programs using an RDBMS.  
FO-Rewritable d1-Programs

- **Basic Idea:**
  - Transform a d1-program \( \Pi = (\mathcal{O}, P) \) into an SQL expression \( S(\Pi) \) over the vocabulary of \( \mathcal{O} \)
  - Desired property: \( S(\Pi) \) is independent of the concrete ABox of \( \mathcal{O} \) management systems (DBMS)
  - To evaluate \( S(\Pi) \), we can use efficient relational database

- The family of DL-Lite DLs satisfies the (analog) property (called FO-reducibility) for CQs.
- For d1-programs, we need restrictions on the rules and the ontology

**Definition**

A d1-program \( \Pi = (\mathcal{O}, P), \mathcal{O} = (\mathcal{T}, \mathcal{A}) \), is **FO-rewritable**, if \( \Pi \models p(\vec{c}) \) for atom \( p(\vec{c}) \), is expressible by a FO formula \( \phi(\vec{x}) \) over the relational schema induced by the vocabulary of \( \mathcal{O} \), such that \( \Pi \models p(\vec{c}) \) iff \( \mathcal{A} \models \phi(\vec{c}) \), where \( \phi \) only depends on \( p, P \) and \( \mathcal{T} \), but not on \( \mathcal{A} \).
Acyclic dl-programs

- To ensure FO-rewritability, ban intrinsic recursion from $\Pi = (\mathcal{O}, P)$
- This is ensured by acyclicity:

  $P$ is acyclic, if some mapping $\mathcal{K}: \text{Preds}(P) \rightarrow \{0, \ldots, n\}$ exists such that for every rule

  $a \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m,$

  in $P$, and every $p, q \in \text{Preds}(P)$ where $p$ occurs in $a$ and $q$ occurs in some $b_i$, it holds $\mathcal{K}(p) > \mathcal{K}(q)$.

Example

$\Pi = (\mathcal{O}, P)$ where $\mathcal{O} = \{C \sqsubseteq D\}$ and

$P = \left\{ p(a); \quad p(b); \quad q(c); \quad s(X) \leftarrow \text{DL}[C \uplus p; D](X), \quad \text{not } \text{DL}[C \uplus q, C \uplus p; D](X) \right\}$.

- Note: every acyclic $\Pi$ has $WFS(\Pi)$ as its unique answer set.
FO-Rewritable dl-atoms

- For FO-rewritability of $\Pi = (\mathcal{O}, P)$, $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ each dl-query $Q(x)$ in $P$ must be FO-rewritable, i.e., some FO-formula $\phi_Q(x)$ on $\mathcal{O}$'s vocabulary exists, such that $\mathcal{O} \models Q(c)$ iff $\mathcal{A} \models \phi_Q(c)$, for each $c$

- $\phi_Q(x)$ must depend only on $\mathcal{T}$, but not on $\mathcal{A}$

Example (cont’d)

dl-query $Q = D(X)$ over $\mathcal{O} = \{ C \sqsubseteq D \}$ is translated to

$$\phi_Q(x) = C(x) \lor D(x)$$

- For dl-atom $DL[\lambda, Q](x)$, also updates $S_i \ op_i \ p_i$ must be respected

Example (cont’d)

The dl-atom $DL[C \sqcup p; D](X)$ is translated into

$$\delta_1(x) = (C(x) \lor p(x)) \lor D(x).$$
FO-rewritable dl-atoms (cont’d)

- if $op_i = \cup$ occurs in $P$, avoid translating $S_i \cup p_i$ to $S_i(x) \lor \neg p_i(x)$

Assume $\mathcal{O}$ is over a DL that is

(i) **CWA-satisfiable** (i.e., for every DL KB $\mathcal{O}'$, the DL KB $\text{CWA}(\mathcal{O}') = \mathcal{O}' \cup \{\neg \alpha \mid \alpha \in A_\Sigma, \mathcal{O}' \not\models \alpha\}$ is satisfiable, where $A_\Sigma$ is the set of all membership assertions in the underlying vocabulary $\Sigma$, and

(ii) allows for FO-rewritable concept and role memberships.

**Example (cont’d)**

The dl-atom $DL[C \cup q; C \cup p; D](X)$ is translated into

$$\delta_2(x) = (C(x) \lor q(x)) \lor D(x) \lor \exists y((C(y) \lor q(y)) \land p(y))$$
FO-Rewritability Result

Theorem ([E_ et al., 2011])

Let $\Pi = (\mathcal{O}, P)$ be acyclic, and $p(\overline{c})$ an atom, such that

1. every dl-query in $P$ is FO-rewritable, and
2. if $\cup$ occurs in $P$, then $\mathcal{O}$ is defined over a DL that (2a) is CWA-satisfiable, and (2b) allows for FO-rewritable concept and role memberships.

Then, $\Pi \models_{wf} p(\overline{c})$ is FO-rewritable.

Constructive proof:

(a) every dl-atom $\delta$ is expressible as FO formula over the ABox of $\mathcal{O}$;
(b) every predicate of rank 0 is easily FO-expressible over the facts of $P$;
(c) every other predicate $p_I$ is expressible by rule merging (cf. Clark Completion)
Example (cont’d)

The rule for predicate $s$ is translated into

$$\phi_s(x) = (\delta_1(x) \land \neg\delta_2(x))$$

Then $\Pi \models_{wf} s(o)$ iff $F \models \phi_s(o)$, for any constant $o$.

Remark:

- The DL-Lite family is CWA-satisfiable
- There, dl-queries $C(X)$, $R(X, Y)$ are immediately FO-rewritable
- Other dl-queries can be reduced to such queries (introducing fresh individuals).
MOR System

**MOR (MergeRuleOntology)** [Schneider, 2010]: experimental prototype

- Evaluates conjunctive queries $CQ$ over an acyclic dl-program $\Pi = (\mathcal{O}, P)$ using an RDBMS (PostgreSQL 8.4)

- Main modules:
  - **Datalog-to-SQL rewriter:**
    puts the facts of $P$ and the ABox of $\mathcal{O}$ in the DB and rewrites the rules of $P$ into cascading VIEWS (not materialized)
  - **DL-Lite plugin:**
    transforms dl-atoms, using the *perfect rewriting* of a query and a TBox $\mathcal{T}$ from the algorithm PerfectRef [Calvanese et al., 2007]
  - **OWLGRES (adapted):**
    construct the *perfect rewriting* (no execution)

- Realize hypothetical updates $S_i \cup p_i$ in dl-atoms by views

- Other plugins than DL-lite are possible (access other DLs, even other formalisms)