Knowledge Representation for the Semantic Web

Lecture 8: Answer Set Programming III

Daria Stepanova

D5: Databases and Information Systems
Max Planck Institute for Informatics

WS 2017/18
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$john \in Person$</td>
<td>person(john)</td>
</tr>
<tr>
<td>$(john, bob) \in hasChild$</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>( john \in Person )</td>
<td>person(john)</td>
</tr>
<tr>
<td>((john, bob) \in hasChild)</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \circ r \sqsubseteq r ) (owl:transitiveProperty)</td>
<td>[ r \circ r \leftarrow r, r ]</td>
</tr>
<tr>
<td>( r \equiv r^- ) (owl:symmetricProperty)</td>
<td>[ r \leftarrow Y, X ]</td>
</tr>
<tr>
<td>( r \equiv s^- ) (owl:symmetricProperty)</td>
<td>[ s \leftarrow Y, X ]</td>
</tr>
<tr>
<td>( C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A )</td>
<td>[ a \leftarrow c_1, \ldots, c_n ]</td>
</tr>
<tr>
<td>( \exists r. C \sqsubseteq A ) (owl:someValuesFrom, lhs)</td>
<td>[ a \leftarrow r, c ]</td>
</tr>
<tr>
<td>( \geq 1 r \sqsubseteq A ) (owl:minCardinality 1, lhs)</td>
<td>[ a \leftarrow r ]</td>
</tr>
<tr>
<td>( A \sqsubseteq \forall r. C ) (owl:allValuesFrom, rhs)</td>
<td>[ a \leftarrow \forall r. c ]</td>
</tr>
<tr>
<td>( A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n )</td>
<td>[ a \leftarrow \forall r. c ]</td>
</tr>
<tr>
<td>( C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A )</td>
<td>[ a \leftarrow c_1, \ldots, c_n ]</td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(john \in \text{Person})</td>
<td>(\text{person}(john))</td>
</tr>
<tr>
<td>((john, bob) \in \text{hasChild})</td>
<td>(\text{hasChild}(john, bob))</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r \circ r \sqsubseteq r) (owl:transitiveProperty)</td>
<td>(r(X, Z) \leftarrow r(X, Y), r(Y, Z)).</td>
</tr>
<tr>
<td>(r \equiv r^\top) (owl:symmetricProperty)</td>
<td></td>
</tr>
<tr>
<td>(r \equiv s^\top) (owl:symmetricProperty)</td>
<td></td>
</tr>
<tr>
<td>(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A)</td>
<td></td>
</tr>
<tr>
<td>(\exists r. C \sqsubseteq A) (owl:someValuesFrom, lhs)</td>
<td></td>
</tr>
<tr>
<td>(\geq 1 r \sqsubseteq A) (owl:minCardinality 1, lhs)</td>
<td></td>
</tr>
<tr>
<td>(A \sqsubseteq \forall r. C) (owl:allValuesFrom, rhs)</td>
<td></td>
</tr>
<tr>
<td>(A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n) (owl:unionOf, rhs)</td>
<td></td>
</tr>
<tr>
<td>(C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A) (owl:unionOf, lhs)</td>
<td></td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(john \in \text{Person})</td>
<td>(\text{person}(john))</td>
</tr>
<tr>
<td>((john, bob) \in \text{hasChild})</td>
<td>(\text{hasChild}(john, bob))</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r \circ r \sqsubseteq r \text{(owl:transitiveProperty)})</td>
<td>(r(X, Z) \leftarrow r(X, Y), r(Y, Z).)</td>
</tr>
<tr>
<td>(r \equiv r^- \text{(owl:symmetricProperty)})</td>
<td>(r(Y, X) \leftarrow r(X, Y).)</td>
</tr>
<tr>
<td>(r \equiv s^- \text{(owl:symmetricProperty)})</td>
<td></td>
</tr>
<tr>
<td>(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A)</td>
<td></td>
</tr>
<tr>
<td>(\exists r.C \sqsubseteq A \text{(owl:someValuesFrom, lhs)})</td>
<td></td>
</tr>
<tr>
<td>(\geq 1 r \sqsubseteq A \text{(owl:minCardinality 1, lhs)})</td>
<td></td>
</tr>
<tr>
<td>(A \sqsubseteq \forall r.C \text{(owl:allValuesFrom, rhs)})</td>
<td></td>
</tr>
<tr>
<td>(A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n \text{(owl:unionOf, rhs)})</td>
<td></td>
</tr>
<tr>
<td>(C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A \text{(owl:unionOf, lhs)})</td>
<td></td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>john (\in) Person</td>
<td>person(john)</td>
</tr>
<tr>
<td>(john, bob) (\in) hasChild</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r \circ r \sqsubseteq r) (owl:transitiveProperty)</td>
<td>(r(X, Z) \leftarrow r(X, Y), r(Y, Z)).</td>
</tr>
<tr>
<td>(r \equiv r^\sim) (owl:symmetricProperty)</td>
<td>(r(Y, X) \leftarrow r(X, Y)).</td>
</tr>
<tr>
<td>(r \equiv s^\sim) (owl:symmetricProperty)</td>
<td>(s(Y, X) \leftarrow r(X, Y)).</td>
</tr>
<tr>
<td>(C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A)</td>
<td></td>
</tr>
<tr>
<td>(\exists r.C \sqsubseteq A) (owl:someValuesFrom, lhs)</td>
<td></td>
</tr>
<tr>
<td>(\geq 1 r \sqsubseteq A) (owl:minCardinality 1, lhs)</td>
<td></td>
</tr>
<tr>
<td>(A \sqsubseteq \forall r.C) (owl:allValuesFrom, rhs)</td>
<td></td>
</tr>
<tr>
<td>(A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n) (owl:unionOf, rhs)</td>
<td></td>
</tr>
<tr>
<td>(C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A) (owl:unionOf, lhs)</td>
<td></td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>john ∈ Person</td>
<td>person(john)</td>
</tr>
<tr>
<td>(john, bob) ∈ hasChild</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \circ r \sqsubseteq r ) (owl:transitiveProperty)</td>
<td>( r(X, Z) \leftarrow r(X, Y), r(Y, Z) ).</td>
</tr>
<tr>
<td>( r \equiv r^- ) (owl:symmetricProperty)</td>
<td>( r(Y, X) \leftarrow r(X, Y) ).</td>
</tr>
<tr>
<td>( r \equiv s^- ) (owl:symmetricProperty)</td>
<td>( s(Y, X) \leftarrow r(X, Y) ).</td>
</tr>
<tr>
<td>( C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A )</td>
<td>( a(X) \leftarrow c_1(X), \ldots, c_n(X) ).</td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>john ∈ Person</td>
<td>person(john)</td>
</tr>
<tr>
<td>(john, bob) ∈ hasChild</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r \circ r \sqsubseteq r$ (owl:transitiveProperty)</td>
<td>$r(X, Z) \leftarrow r(X, Y), r(Y, Z)$.</td>
</tr>
<tr>
<td>$r \equiv r^-$ (owl:symmetricProperty)</td>
<td>$r(Y, X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$r \equiv s^-$ (owl:symmetricProperty)</td>
<td>$s(Y, X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$</td>
<td>$a(X) \leftarrow c_1(X), \ldots, c_n(X)$.</td>
</tr>
<tr>
<td>$\exists r.C \sqsubseteq A$ (owl:someValuesFrom, lhs)</td>
<td>$a(X) \leftarrow r(X, Y), c(Y)$.</td>
</tr>
<tr>
<td>$\geq 1 r \sqsubseteq A$ (owl:minCardinality 1, lhs)</td>
<td></td>
</tr>
<tr>
<td>$A \sqsubseteq \forall r.C$ (owl:allValuesFrom, rhs)</td>
<td></td>
</tr>
<tr>
<td>$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf, rhs)</td>
<td></td>
</tr>
<tr>
<td>$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf, lhs)</td>
<td></td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>john ∈ Person</td>
<td>person(john)</td>
</tr>
<tr>
<td>(john, bob) ∈ hasChild</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r \circ r \sqsubseteq r$ (owl:transitiveProperty)</td>
<td>$r(X, Z) \leftarrow r(X, Y), r(Y, Z)$.</td>
</tr>
<tr>
<td>$r \equiv r^-$ (owl:symmetricProperty)</td>
<td>$r(Y, X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$r \equiv s^-$ (owl:symmetricProperty)</td>
<td>$s(Y, X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$</td>
<td>$a(X) \leftarrow c_1(X), \ldots, c_n(X)$.</td>
</tr>
<tr>
<td>$\exists r.C \sqsubseteq A$ (owl:someValuesFrom, lhs)</td>
<td>$a(X) \leftarrow r(X, Y), c(Y)$.</td>
</tr>
<tr>
<td>$\geq 1 r \sqsubseteq A$ (owl:minCardinality 1, lhs)</td>
<td>$a(X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$A \sqsubseteq \forall r.C$ (owl:allValuesFrom, rhs)</td>
<td></td>
</tr>
<tr>
<td>$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf, rhs)</td>
<td></td>
</tr>
<tr>
<td>$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf, lhs)</td>
<td></td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>john ∈ Person</td>
<td>person(john)</td>
</tr>
<tr>
<td>(john, bob) ∈ hasChild</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \circ r \sqsubseteq r ) (owl:transitiveProperty)</td>
<td>( r(X, Z) \leftarrow r(X, Y), r(Y, Z) ).</td>
</tr>
<tr>
<td>( r \equiv r^- ) (owl:symmetricProperty)</td>
<td>( r(Y, X) \leftarrow r(X, Y) ).</td>
</tr>
<tr>
<td>( r \equiv s^- ) (owl:symmetricProperty)</td>
<td>( s(Y, X) \leftarrow r(X, Y) ).</td>
</tr>
<tr>
<td>( C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A )</td>
<td>( a(X) \leftarrow c_1(X), \ldots, c_n(X) ).</td>
</tr>
<tr>
<td>( \exists r.C \sqsubseteq A ) (owl:someValuesFrom, lhs)</td>
<td>( a(X) \leftarrow r(X, Y), c(Y) ).</td>
</tr>
<tr>
<td>( \geq 1 r \sqsubseteq A ) (owl:minCardinality 1, lhs)</td>
<td>( a(X) \leftarrow r(X, Y) ).</td>
</tr>
<tr>
<td>( A \sqsubseteq \forall r.C ) (owl:allValuesFrom, rhs)</td>
<td>( c(Y) \leftarrow r(X, Y), a(X) ).</td>
</tr>
<tr>
<td>( A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n ) (owl:unionOf, rhs)</td>
<td></td>
</tr>
<tr>
<td>( C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A ) (owl:unionOf, lhs)</td>
<td></td>
</tr>
</tbody>
</table>
What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>John ∈ Person</td>
<td>person(john)</td>
</tr>
<tr>
<td>(john, bob) ∈ hasChild</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r \circ r \sqsubseteq r$ (owl:transitiveProperty)</td>
<td>$r(X, Z) \leftarrow r(X, Y), r(Y, Z)$.</td>
</tr>
<tr>
<td>$r \equiv r^-$ (owl:symmetricProperty)</td>
<td>$r(Y, X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$r \equiv s^-$ (owl:symmetricProperty)</td>
<td>$s(Y, X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$</td>
<td>$a(X) \leftarrow c_1(X), \ldots, c_n(X)$.</td>
</tr>
<tr>
<td>$\exists r.C \sqsubseteq A$ (owl:someValuesFrom, lhs)</td>
<td>$a(X) \leftarrow r(X, Y), c(Y)$.</td>
</tr>
<tr>
<td>$\geq 1 r \sqsubseteq A$ (owl:minCardinality 1, lhs)</td>
<td>$a(X) \leftarrow r(X, Y)$.</td>
</tr>
<tr>
<td>$A \sqsubseteq \forall r.C$ (owl:allValuesFrom, rhs)</td>
<td>$c(Y) \leftarrow r(X, Y), a(X)$.</td>
</tr>
<tr>
<td>$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf, rhs)</td>
<td>$c_1(X) \lor \ldots \lor c_n(X) \leftarrow a(X)$.</td>
</tr>
</tbody>
</table>
### What of DLs can be expressed directly in ASP?

- **ABox**: Factual knowledge about class membership and property values can be translated to ASP “as is”:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>john ∈ Person</td>
<td>person(john)</td>
</tr>
<tr>
<td>(john, bob) ∈ hasChild</td>
<td>hasChild(john, bob)</td>
</tr>
</tbody>
</table>

- **RBox/TBox**: A subset of OWL can be straightforwardly translated to ASP, here only a subset is given:

<table>
<thead>
<tr>
<th>DL syntax</th>
<th>ASP syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r_1 o r_1 ⊑ r_1</code> (owl:transitiveProperty)</td>
<td><code>r(X, Z) ← r(X, Y), r(Y, Z).</code></td>
</tr>
<tr>
<td><code>r_1 ≡ r_1</code> (owl:symmetricProperty)</td>
<td><code>r(Y, X) ← r(X, Y).</code></td>
</tr>
<tr>
<td><code>r_1 ≡ s_1</code> (owl:symmetricProperty)</td>
<td><code>s(Y, X) ← r(X, Y).</code></td>
</tr>
<tr>
<td><code>C_1 ⊓ ... ⊓ C_n ⊑ A</code></td>
<td><code>a(X) ← c_1(X), ..., c_n(X).</code></td>
</tr>
<tr>
<td><code>∃ r . C ⊑ A</code> (owl:someValuesFrom, lhs)</td>
<td><code>a(X) ← r(X, Y), c(Y).</code></td>
</tr>
<tr>
<td><code>≥ 1 r ⊑ A</code> (owl:minCardinality 1, lhs)</td>
<td><code>a(X) ← r(X, Y).</code></td>
</tr>
<tr>
<td><code>A ⊑ ∀ r . C</code> (owl:allValuesFrom, rhs)</td>
<td><code>c(Y) ← r(X, Y), a(X).</code></td>
</tr>
<tr>
<td><code>A ⊑ C_1 ⊔ ... ⊔ C_n</code> (owl:unionOf, rhs)</td>
<td><code>c_1(X) ∨ ... ∨ c_n(X) ← a(X).</code></td>
</tr>
<tr>
<td><code>C_1 ⊔ ... ⊔ C_n ⊑ A</code></td>
<td><code>a(X) ← C_1(X). ... A(X) ← C_n(X).</code></td>
</tr>
</tbody>
</table>
What of DLs cannot be directly expressed in ASP?

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \equiv {o_1, \ldots, o_n}$ (owlOneOf)</td>
<td>Cannot be directly translated</td>
</tr>
<tr>
<td>$A \subseteq \exists r.C$</td>
<td>Impossible to express as there, no existentials in the heads</td>
</tr>
<tr>
<td>$\forall r.C \subseteq A$</td>
<td>One might guess:</td>
</tr>
<tr>
<td></td>
<td>$a(X) \leftarrow \text{not no}_rc(X)$.</td>
</tr>
<tr>
<td></td>
<td>$\text{no}_rc(X) \leftarrow r(X, Y), \neg c(Y)$.</td>
</tr>
<tr>
<td></td>
<td>but does not work.</td>
</tr>
<tr>
<td></td>
<td><strong>Exercise: Why?</strong></td>
</tr>
</tbody>
</table>
Main difference between DLs and ASP

- ¬ in DLs is different from *not* in LP
  - ¬: classical negation, monotonicity, open world assumption
  - *not*: default negation, nonmonotonicity, closed world assumption

<table>
<thead>
<tr>
<th>DL ontology ( \mathcal{K} )</th>
<th>ASP Program ( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Child \sqsubseteq Person )</td>
<td>( \text{person}(X) \leftarrow \text{child}(X) )</td>
</tr>
<tr>
<td>( \neg Child \sqsubseteq \text{Adult} )</td>
<td>( \text{adult}(X) \leftarrow \neg \text{child}(X) )</td>
</tr>
<tr>
<td>( \text{Person}(\text{john}) )</td>
<td>( \text{person}(\text{john}) )</td>
</tr>
<tr>
<td>( \mathcal{K} \not\models \text{Adult}(\text{john}) )</td>
<td>( P ) infers ( \text{adult}(\text{john}) )</td>
</tr>
</tbody>
</table>

- DLs are strong in subsumption checking, LPs in expressing relations
- DLs allow complex expressions in heads (rhs of \( \sqsubseteq \)), while in LPs use of variables in rule bodies is more flexible
- ...
DLs vs ASP

DL Ontologies

- Open-World Assumption
- Monotonic
- Conceptual reasoning

ASP Rules

- Closed-World Assumption
- Nonmonotonic
- Defaults and exceptions
DLs vs ASP

Hybrid Knowledge Bases

MKNF, DL-safe rules, **DL-programs**...

**DL Ontologies**
- Open-World Assumption
- Monotonic
- Conceptual reasoning

**ASP Rules**
- Closed-World Assumption
- Nonmonotonic
- Defaults and exceptions
DL-programs
DL-programs: loose coupling of DL ontologies and ASP rules [Eiter et al., 2008]
### DL-program

#### DL ontology

<table>
<thead>
<tr>
<th>TBox</th>
<th>ABox (KG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Child ⊑ ∃ hasParent</td>
<td>(4) Male(pat)</td>
</tr>
<tr>
<td>(2) Female ⊑ ¬ Male</td>
<td>(5) Male(john)</td>
</tr>
<tr>
<td>(3) Adopted ⊑ Child</td>
<td>(6) hasParent(john, pat)</td>
</tr>
</tbody>
</table>

#### Rules

(7) isChildOf(john, alex).
(8) boy(tim).
(9) hasFather(john, pat) ← 📞, 📞
## DL-program

### DL ontology

#### TBox

1. \( \text{Child} \sqsubseteq \exists \text{hasParent} \)
2. \( \text{Female} \sqsubseteq \neg \text{Male} \)
3. \( \text{Adopted} \sqsubseteq \text{Child} \)

#### ABox (KG)

4. \( \text{Male}(\text{pat}) \)
5. \( \text{Male}(\text{john}) \)
6. \( \text{hasParent}(\text{john}, \text{pat}) \)

### Rules

7. \( \text{isChildOf}(\text{john}, \text{alex}). \)
8. \( \text{boy}(\text{tim}). \)

9. \( \text{hasFather}(\text{john}, \text{pat}) \leftarrow \text{DL}[; \text{hasParent}](\text{john}, \text{pat}), \)
**DL-program**

### DL ontology

#### TBox

1. \( \text{Child} \sqsubseteq \exists \text{hasParent} \)
2. \( \text{Female} \sqsubseteq \neg \text{Male} \)
3. \( \text{Adopted} \sqsubseteq \text{Child} \)

#### ABox (KG)

4. \( \text{Male(} \text{pat} \text{)} \)
5. \( \text{Male(} \text{john} \text{)} \)
6. \( \text{hasParent(} \text{john}, \text{pat} \text{)} \)

### Rules

7. \( \text{isChildOf(} \text{john}, \text{alex} \text{)} \).
8. \( \text{boy(} \text{tim} \text{)} \).
9. \( \text{hasFather}( \text{john}, \text{pat} ) \leftarrow \text{DL}[; \text{hasParent}( \text{john}, \text{pat} )] \)
# DL-program

## DL ontology

<table>
<thead>
<tr>
<th>TBox</th>
<th>ABox (KG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Child $\sqsubseteq \exists$ hasParent</td>
<td>(4) Male(pat)</td>
</tr>
<tr>
<td>(2) Female $\sqsubseteq \neg$ Male</td>
<td>(5) Male(john)</td>
</tr>
<tr>
<td>(3) Adopted $\sqsubseteq$ Child</td>
<td>(6) hasParent(john, pat)</td>
</tr>
</tbody>
</table>

## Rules

(7) isChildOf(john, alex).  
(8) boy(tim).
(9) hasFather(john, pat) $\leftarrow$ DL[; hasParent](john, pat),  
DL[Male $\uplus$ boy; Male](pat)
# DL-program

## DL ontology

<table>
<thead>
<tr>
<th>TBox</th>
<th>ABox (KG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) ( \text{Child} \sqsubseteq \exists \text{hasParent} )</td>
<td>(4) ( \text{Male}(\text{pat}) )</td>
</tr>
<tr>
<td>(2) ( \text{Female} \sqsubseteq \neg \text{Male} )</td>
<td>(5) ( \text{Male}(\text{john}) )</td>
</tr>
<tr>
<td>(3) ( \text{ Adopted} \sqsubseteq \text{Child} )</td>
<td>(6) ( \text{hasParent}(\text{john}, \text{pat}) )</td>
</tr>
</tbody>
</table>

## Rules

(7) \( \text{isChildOf}(\text{john}, \text{alex}). \)  (8) \( \text{boy}(\text{tim}). \)

(9) \( \text{hasFather}(\text{john}, \text{pat}) \leftarrow \text{DL[} ; \text{hasParent}(\text{john}, \text{pat})\text{]}. \) \( , \) \( \text{DL[} \text{Male} \uplus \text{boy} ; \text{Male}] (\text{pat}) \)
## DL-program

### DL ontology

<table>
<thead>
<tr>
<th>TBox</th>
<th>ABox (KG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Child ⊆ ∃ hasParent</td>
<td>(4) Male(pat)</td>
</tr>
<tr>
<td>(2) Female ⊆ ¬ Male</td>
<td>(5) Male(john)</td>
</tr>
<tr>
<td>(3) Adopted ⊆ Child</td>
<td>(6) hasParent(john, pat)</td>
</tr>
</tbody>
</table>

### Rules

(7) `isChildOf(john, alex)`.  
(8) `boy(tim)`.  
(9) `hasFather(john, pat) ← DL[; hasParent](john, pat)`,  
    DL[Male ⊔ boy; Male](pat)
## DL-program

### DL ontology

<table>
<thead>
<tr>
<th>TBox</th>
<th>ABox (KG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Child $\sqsubseteq \exists \text{hasParent}$</td>
<td>(4) Male($pat$) Male($tim$)</td>
</tr>
<tr>
<td>(2) Female $\sqsubseteq \neg \text{Male}$</td>
<td>(5) Male($john$)</td>
</tr>
<tr>
<td>(3) Adopted $\sqsubseteq \text{Child}$</td>
<td>(6) hasParent($john$, $pat$)</td>
</tr>
</tbody>
</table>

### Rules

(7) isChildOf($john$, $alex$)  
(8) boy($tim$)  
(9) hasFather($john$, $pat$) $\leftarrow$ DL[$;$ hasParent]($john$, $pat$),  
DL[Male $\sqcup$ boy; Male]($pat$) $\checkmark$
### DL-program

#### DL ontology

**TBox**

1. \( \text{Child} \sqsubseteq \exists \text{hasParent} \)
2. \( \text{Female} \sqsubseteq \lnot \text{Male} \)
3. \( \text{Adopted} \sqsubseteq \text{Child} \)

**ABox (KG)**

4. \( \text{Male}(\text{pat}) \)
5. \( \text{Male}(\text{john}) \)
6. \( \text{hasParent}(\text{john}, \text{pat}) \)

#### Rules

7. \( \text{isChildOf}(\text{john}, \text{alex}) \).
8. \( \text{boy}(\text{tim}) \).
9. \( \text{hasFather}(\text{john}, \text{pat}) \leftarrow \text{DL}[; \text{hasParent}](\text{john}, \text{pat}), \)
\( \text{DL}[	ext{Male} \uplus \text{boy}; \text{Male}](\text{pat}) \)

**Answer set:** \( I = \{ \text{isChildOf}(\text{john}, \text{alex}), \text{boy}(\text{tim}), \text{hasFather}(\text{john}, \text{pat}) \} \)
Example: Semantic Route Planning

- Personalized semantic route planning
- Requirements:
  - Find shortest trip visiting predefined locations
  - If the shortest trip is beyond a certain length, add lunch location satisfying user’s preferences
- DL ontology: restaurant classification
- ASP rules: optimal path with user constraints
Other ASP Extensions
Extensions of ASP

Language extensions like aggregates, complex formula syntax are within same semantic / computational framework

**Need:**

- interoperability with other logics
- interfacing with programming languages, e.g. C++, Python
- access to general *external* sources of information, e.g. WordNet

**Approaches:**

- ASP + concrete theories, e.g.,
  - ASP + DL ontologies (DL-programs\(^1\))
  - constraint ASP
- ASP + abstract theories, e.g.,
  - HEX-programs\(^2\)

---

\(^1\)https://github.com/hexhex/dlliteplugin
\(^2\)http://www.kr.tuwien.ac.at/research/systems/dlvhex/
External Information Access

Examples:

- Import external knowledge graph triples into the program:
  \[ (S, P, O) \leftarrow \text{\&rdf} \left[ "http://\langle Nick\rangle.livejournal.com/data/foaf" \right] \]
  \[ (S, P, O) \]

- Access external graph reachable:
  \[ X \leftarrow \text{\&reachable} \left[ \text{conn}, a \right] \]

- Perform auxiliary / data structure computations:
  \[ \text{fullname}(Z) \leftarrow \text{\& concat} \left[ X, Y \right] \]
  \[ \text{firstname}(X), \text{lastname}(Y) \]
External Information Access

Examples:

- import external knowledge graph triples into the program

\[
\text{triple}(S, P, O) \leftarrow \&\text{rdf}["http://\langle Nick\rangle.livejournal.com/data/foaf"](S, P, O).
\]
External Information Access

Examples:

- import external knowledge graph triples into the program
  \[\text{triple}(S, P, O) \leftarrow &\text{rdf}[^{http://\langle Nick \rangle.livejournal.com/data/foaf}](S, P, O)\].

- access external graph
  \[\text{reachable}(X) \leftarrow &\text{reachable}[\text{conn, a}](X)\].
External Information Access

Examples:

- import external knowledge graph triples into the program
  \[
  \text{triple}(S, P, O) \leftarrow \text{\&rdf["http://\langle \text{Nick} \rangle.livejournal.com/data/foaf"]}(S, P, O).
  \]

- access external graph
  \[
  \text{reachable}(X) \leftarrow \text{\&reachable[\text{conn, a}]}(X).
  \]

- perform auxiliary / data structure computations
  \[
  \text{fullname}(Z) \leftarrow \text{\&concat}[X, Y](Z), \text{firstname}(X), \text{lastname}(Y).
  \]
Examples: External Atoms

Concatenate two strings:

- $\&\text{concat}[X, Y](Z)$: intuitively, concatenate two strings
  - $\&\text{concat}[\text{bob}, \text{dylan}](\text{bobdylan})$ is true
  - $\&\text{concat}[\text{bob}, \text{dylan}](Z)$ is true for $Z = \text{bobdylan}$
  - $\&\text{concat}[\text{bob}, Y](\text{bobdylan})$ is true for $Y = \text{dylan}$
Examples: External Atoms (cont’d)

Query a web-based weather report:

- \&weatherreport[dateLocationPredicate](WeatherConditions)

  - input \textit{dateLocationPredicate} is a binary predicate with tuples \((d, l)\) of dates \(d\) and locations \(l\) (facts \textit{dateLocationPredicate}(d, l))
  - output \textit{WeatherConditions} are (one by one) all weather conditions that occur at some input date & location
Example: City Trip

Plan to visit Paris and London, under the condition the weather isn’t bad:

% Define bad weather conditions
(1) badweather(rain).
(2) badweather(snow).

% Decide where to go on which day
(3) goto(1, paris) ∨ goto(1, london).
(4) goto(2, paris) ∨ goto(2, london).

% Rule out invalid trips
(5) ← &weatherreport[goto](W), badweather(W).
Example: AI Agent for Angrybirds Game

- **AngryBirds** is a game, whose goal is to kill pigs with birds
- AI competition to automate playing (angrybirds.org)

  - **Approach:** design an agent based on **declarative** logic programming
    - **Challenge:** plan optimal shots under consideration of some physics

  - **Means:** **HEX-programs**

---

\(^a\) Joint project with TU Wien and University of Calabria
External Atoms Examples

- **HEX-program** $\Pi$ for shot computation
- Examples of external atoms:
  - $\&\text{distance}[O_1, O_2](D)$ is true iff distance between $O_1$ and $O_2$ is $D$
  - $\&\text{canpush}[\text{ngobject}](O_1, O_2)$ is true iff $O_1$ can push $O_2$ given additional info in the extention of $\text{ngobject}$

- **Rule** $1$ estimates the likelihood that object $O_2$ falls when $O_1$ is hit

**Rule** $1$: $\text{pushDamage}(O_2, P_1, P) \leftarrow \text{pushDamage}(O_1, -, P_1), P_1 > 0$

$\&\text{canpush}[\text{ngobject}](O_1, O_2)$,

$\text{pushability}(O_2, P_2), P = P_1 \times P_2 / 100.$
Architecture of Angry-HEX

- We use the provided framework (browser plugin, vision module, . . .)
- Agent builds on Tactics and Strategy, both are realized declaratively

- **Tactics:** reasoning about the next shot is done in a HEX-program \( \Pi \)
  - **Input:** scene info from the vision module (facts of \( \Pi \))
  - **Output:** desired target (models of \( \Pi \))

- **Strategy:** next level to be played is computed in an ASP program \( \Pi' \)
  - **Input:** info about the number of times levels were played, best scores achieved, scores of our agent (facts of \( \Pi' \))
  - **Output:** next optimal level to be played (models of \( \Pi' \))
**HEX Encoding for Tactics**

- **Physics simulation results** are accessed via external atoms:
  - decide which \( O' \) intersect with trajectory of a bird after hitting \( O \)
  - decide whether \( O_1 \) falls whenever \( O_2 \) falls . . .

- **Tactics in details:**
  - Consider each shootable target
  - Compute the estimated damage on each non-target object
  - Rank the targets (=Answer Sets) using weak constraints
  - Consider history: never play a level in the same way again!
ASP Encoding for Strategy

- **Decides which level to play next** based on info about:
  - number of times each level was played
  - best scores
  - our agent’s scores . . .

- **Strategy in details:**
  - **First** play each level once
  - **Second** play levels in which our score maximally differs from the best one
  - **Third** play levels in which we played best and the difference to the second best score is minimal
Rule Learning
Motivation

ASP programs are usually constructed by domain experts

**Issue:** requires a lot of manual efforts!

**Question:** Can we learn rules from data, e.g., from knowledge graphs?
Horn Rule Mining

Conf\( (r) = |livesIn| + |livesIn| = 2\)

\[livesIn(\text{Brad}, \text{Berlin}) \leftarrow \text{isMarriedTo}(\text{Bob}, \text{Alice})\]

\[livesIn(\text{Ann}, \text{Berlin}) \leftarrow \text{isMarriedTo}(\text{Brad}, \text{Ann})\]

\[livesIn(\text{John}, \text{Chicago}) \leftarrow \text{isMarriedTo}(\text{Dave}, \text{Clara})\]

\[livesIn(\text{Kate}, \text{Chicago}) \leftarrow \text{hasBrother}(\text{John}, \text{Kate})\]
Horn Rule Mining

Horn rule mining for KG completion [Galárraga et al., 2015]

conf(r) = \frac{|\triangle|}{|\triangle| + |\bowtie|} = \frac{2}{4}

r : \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z)
Horn Rule Mining

Horn rule mining for KG completion [Galárraga et al., 2015]

\[
\text{conf}(r) = \frac{\mid \triangledown \mid}{\mid \triangledown \mid + \mid \bigcirc \mid} = \frac{2}{4}
\]

\[ r : \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z) \]
Nonmonotonic Rule Mining

Nonmonotonic rule mining from KGs: OWA is a challenge!

\[ \text{conf}(r) = \frac{|\Delta|}{|\Delta| + |\Lambda|} = 1 \]

\( r : \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \neg \text{researcher}(X) \)
Declarative Programming Paradigm

PROBLEM

Modeling

NONMONOTONIC RULES

SOLVING

SOLVING

ASP solvers, e.g. clingo, dlv, dlvhex...

SOLVING

INTERPRETING

SOLUTION

ANSWER SET
Declarative Programming Example

Graph 3-colorability

Modeling

\[
\begin{align*}
\text{node}(1 \ldots 6) &; \quad \text{edge}(1, 2); \quad \ldots \\
\text{col}(V, \text{red}) & \leftarrow \neg \text{col}(V, \text{blue}), \neg \text{col}(V, \text{green}), \text{node}(V); \\
\text{col}(V, \text{green}) & \leftarrow \neg \text{col}(V, \text{blue}), \neg \text{col}(V, \text{red}), \text{node}(V); \\
\text{col}(V, \text{blue}) & \leftarrow \neg \text{col}(V, \text{green}), \neg \text{col}(V, \text{red}), \text{node}(V); \\
\bot & \leftarrow \text{col}(V, C), \text{col}(V, C'), C \neq C'; \\
\bot & \leftarrow \text{col}(V, C), \text{col}(V', C), \text{edge}(V, V')
\end{align*}
\]

Interpreting

Solving

\[
\begin{align*}
\text{node}(1 \ldots 6) &; \quad \text{edge}(1, 2); \quad \ldots \\
\text{col}(1, \text{red}), \text{col}(2, \text{blue}), \\
\text{col}(3, \text{red}), \text{col}(4, \text{green}), \\
\text{col}(6, \text{green}), \text{col}(5, \text{blue})
\end{align*}
\]
Nonmonotonic Rule Mining

Knowledge Graph

Partial since KG is incomplete (OWA)

Mining

Partial Answer Set

Nonmonotonic Rules
Nonmonotonic Rule Mining

livesIn(Y, Z) ← isMarried(X, Y), livesIn(X, Y), not researcher(Y)

isMarriedTo(brad, ann); isMarriedTo(john, kate); isMarriedTo(bob, alice); isMarriedTo(clara, dave); livesIn(brad, berlin); . . . researcher(alice); researcher(dave)
Nonmonotonic Rule Mining from KGs

**Goal:** learn nonmonotonic rules from KG

**Approach:** revise association rules learned using data mining methods

Learning rules from data

Inductive logic programming
[Muggleton, 1990]

Learning from examples

Abduction in logic programs
[Eiter et al., 1992, Kakas et al., 2002]

First-order theory refinement
[Wrobel, 1996]

Learning from interpretations
[Law et al., 2014]

Data mining

Association rule learning
[Agrawal et al., 1993]

Clustering

Relational
[Goethals et al., 2002, Galárraga et al., 2015]

Learning nonmonotonic rules from KGs
Horn Theory Revision

Quality-based Horn Theory Revision

Given:
- Available KG

Ideal KG (unknown)

Available KG
Horn Theory Revision

Quality-based Horn Theory Revision

Given:

- Available KG
- Horn rule set
Horn Theory Revision

Quality-based Horn Theory Revision

Given:
- Available KG
- Horn rule set

Find:
- Nonmonotonic revision of Horn rule set
Horn Theory Revision

Quality-based Horn Theory Revision

Given:
- Available KG
- Horn rule set

Find:
- Nonmonotonic revision of Horn rule set with better predictive quality

D. Tran, D. Stepanova, M. Gad-Elrab, F. Lisi, G. Weikum. Towards Nonmonotonic Relational Learning from KGs. ILP2016
Experimental Setup

- **Approximated ideal KG**: original KG
- **Available KG**: for every relation randomly remove 20% of facts from approximated ideal KG
- **Horn rules**: \( h(X, Y) \leftarrow p(X, Z), q(Z, Y) \)
- **Exceptions**: \( e_1(X), e_2(Y), e_3(X, Y) \)
- **Predictions** are computed using answer set solver DLV

Correctly removed false predictions:
- IMDB: 57.75%
- YAGO: 85%
Experimental Setup

- **Approximated ideal KG**: original KG
- **Available KG**: for every relation randomly remove 20% of facts from approximated ideal KG
- **Horn rules**: $h(X, Y) \leftarrow p(X, Z), q(Z, Y)$
- **Exceptions**: $e_1(X), e_2(Y), e_3(X, Y)$
- **Predictions** are computed using answer set solver DLV

**Examples of revised rules**:

Plots of films in a sequel are written by the same writer, unless a film is American

$r_1 : writtenBy(X, Z) \leftarrow hasPredecessor(X, Y), writtenBy(Y, Z), \text{not } american\_film(X)$

Spouses of film directors appear on the cast, unless they are silent film actors

$r_2 : actedIn(X, Z) \leftarrow isMarriedTo(X, Y), directed(Y, Z), \text{not } silent\_film\_actor(X)$
Spurious Rules due to Incompleteness

\[ conf(r) = \frac{|\bigtriangleup|}{|\bigtriangleup| + |\bigtriangledown|} = \frac{2}{3} \]

\( r : \text{isPoliticianOf}(X, Z) \leftarrow \text{hasChild}(X, Y), \text{isCitizenOf}(Y, Z) \)
Spurious Rules due to Incompleteness

In real world:

\[
\text{conf}(r) = \frac{|\triangledown|}{|\triangledown| + |\triangleright|} = \frac{2}{3}
\]

\[r : \text{isPoliticianOf}(X, Z) \leftarrow \text{hasChild}(X, Y), \text{isCitizenOf}(Y, Z)\]
Spurious Rules due to Incompleteness

In real world:

\[
\text{conf}(r) = \frac{|\triangle|}{|\triangle| + |\blacksquare|} = \frac{2}{6}
\]

\(r: \text{isPoliticianOf}(X, Z) \leftarrow \text{hasChild}(X, Y), \text{isCitizenOf}(Y, Z)\)
Completeness-aware Rule Learning

- Exploit cardinality meta-data [Mirza et al., 2016] in rule mining

*John has 5 children, Mary is a citizen of 2 countries*

T. Pellissier-Tanon, D. Stepanova, S. Razniewski, P. Mirza, G. Weikum, Completeness-aware Rule Learning from Knowledge Graphs, ISWC 2017
Research Topics

Reasoning:
- Theoretical and practical open problems both in ASP and DLs
- Hybrid combinations of ASP with other logics
- Applications, e.g., constraint-based frequent pattern discovery\(^3\)

Learning:
- Learn various types of rules from KGs, e.g.,
  - with cardinalities:
    “If you have X siblings, then your parents have X+1 children”
  - with existentials:
    “If you are a famous actor from Hollywood, there is likely an award that you were nominated for“
  - etc.

Hybrid topics:
- Use rules for NLP and information extraction tasks
- Angryhex project

\(^3\) S. Paromonov, D. Stepanova, P. Miettinen, Hybrid ASP-based Approach to Pattern Mining, RR 2017
Summary

- **DIs vs ASP**
  - What of DIs can be expressed in ASP
  - What of DIs cannot be expressed in ASP

- **DL-programs**
  - Example applications

- **HEX-programs**
  - Example applications

- **Rule learning**
  - Rules with exceptions
  - Completeness awareness
Conclusion

Main part:

1. Description Logic ontologies (DL)
   - Syntax and semantics
   - Reasoning problems
   - Ontology Web Language (OWL)
   - Tools and applications

2. Answer Set Programming rules (ASP)
   - Syntax and semantics
   - Declarative programming paradigm
   - Tools and applications

Advanced topics:

3. ASP extensions and rule learning
   - DL-programs
   - HEX-programs
   - Applications
   - Approaches to rule learning


Avoid Data Overfitting

How to distinguish exceptions from noise?

\[ r1 : \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \text{not} \ \text{researcher}(X) \]
Avoid Data Overfitting

How to distinguish exceptions from noise?

\[ r1 : \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \text{not researcher}(X) \]
\[ \text{not livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \text{researcher}(X) \]
Avoid Data Overfitting

How to distinguish exceptions from noise?

\[ r_1 : \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \text{not researcher}(X) \]
\[ \text{not livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \text{researcher}(X) \]

\[ r_2 : \text{livesIn}(X, Z) \leftarrow \text{bornIn}(X, Z), \text{not moved}(X) \]
\[ \text{not livesIn}(X, Z) \leftarrow \text{bornIn}(X, Z), \text{moved}(X) \]

Intuition: Rules with good exceptions should make few conflicting predictions.
Avoid Data Overfitting

How to distinguish exceptions from noise?

\[ r1: \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \text{not researcher}(X) \]
\[ \text{not livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z), \text{researcher}(X) \]

\[ r2: \text{livesIn}(X, Z) \leftarrow \text{bornIn}(X, Z), \text{not moved}(X) \]
\[ \text{not livesIn}(X, Z) \leftarrow \text{bornIn}(X, Z), \text{moved}(X) \]

\{ \text{livesIn}(c, d), \text{not livesIn}(c, d) \} \text{ are conflicting predictions} \]

**Intuition:** Rules with good exceptions should make few conflicting predictions
Horn Theory Revision

Quality-based Horn Theory Revision

**Given:**
- Available KG
- Horn rule set

**Find:**
- Nonmonotonic revision of Horn rules, such that
  - number of conflicting predictions is **minimal**
  - average conviction is **maximal**
Exception Candidates

\[ r: \text{livesIn}(X, Z) \leftarrow \text{isMarriedTo}(Y, X), \text{livesIn}(Y, Z) \]

\[ \{ \text{not researcher}(X) \} \]

\[ \{ \text{not artist}(Y) \} \]
Exception Ranking

rule1 \{e_1, e_2, e_3, \ldots \}
rule2 \{e_1, e_2, e_3, \ldots \}
rule3 \{e_1, e_2, e_3, \ldots \}

Finding globally best revision is expensive, exponentially many candidates!

- **Naive ranking**: for every rule inject exception that results in the highest conviction
- **Partial materialization (PM)**: apply all rules apart from a given one, inject exception that results in the highest average conviction of the rule and its rewriting
- **Ordered PM (OPM)**: same as PM plus ordered rules application
- **Weighted OPM**: same as OPM plus weights on predictions
Hybrid Constraint-based Pattern Mining

- Interlink **mining** and **reasoning**
- Use declarative **logic programming** for frequent pattern (itemset/sequence) filtering
- Combine various **domain-specific** constraints
Semantically-enhanced Fact Spotting

**KG population problem:** some facts are hard to spot in text due to reporting bias $\text{lost}(\text{nadal}, \text{australianOpen2017})$

**Given:**
- Fact: $\text{lost}(\text{nadal}, \text{australianOpen2017})$
- Rule set: $\text{lost}(Z, Y) \leftarrow \text{won}(X, Y), \text{finalist}(Z, Y), X \neq Z$
- KG: $\text{won}(\text{federer}, \text{australianOpen2017})$
- Text: “... another **finalist** of Australian Open in 2017 was Nadal”

**Find:**
- Fact’s truth value: $\text{lost}(\text{nadal}, \text{australianOpen2017})$ is true!

Joint work with Mohamed Gad Elrab, Jacopo Urbani and Gerhard Weikum