A Logical Approach to Privacy-Aware Access to Data

Dennis Fassbender
dfabender@uni-koblenz.de

Uni Koblenz

July 14, 2009
Introduction

Why do we need access control?

► We want to have one knowledge base that is accessed by all users, but...
► We want to prevent different users from retrieving different kinds of information
► Method to define who is allowed to access what information is needed
Introduction

Why do we need access control?

- We want to have one knowledge base that is accessed by all users, but...
- We want to prevent different users from retrieving different kinds of information
- Method to define who is allowed to access what information is needed

Goals of this presentation

- Create Description Logic ontology representing a simple social network
- Use views to control access and enforce privacy policy
Access Control on the Semantic Web

Questions

▶ What are views?
▶ What is an ontology?
▶ What is Description Logic?
▶ How can views be used to restrict access to Description Logic ontologies?
Access Control on the Semantic Web

Questions

▶ What are views?
▶ What is an ontology?
▶ What is Description Logic?
▶ How can views be used to restrict access to Description Logic ontologies?

Views

▶ Traditionally used in relational database world
▶ Filter data that user can access
▶ Can combine data from different relations
▶ Can be used to hide sensitive data from user
### Views - Example

#### users

<table>
<thead>
<tr>
<th>id</th>
<th>nick</th>
<th>password</th>
<th>real_name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>john</td>
<td>e64f6...</td>
<td>John Meyer</td>
<td>Austin, TX</td>
</tr>
<tr>
<td>2</td>
<td>kate</td>
<td>a2cd5...</td>
<td>Kate Stevenson</td>
<td>Miami, FL</td>
</tr>
</tbody>
</table>

#### users_public

<table>
<thead>
<tr>
<th>id</th>
<th>nick</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>john</td>
</tr>
<tr>
<td>2</td>
<td>kate</td>
</tr>
</tbody>
</table>
What is an ontology?

Ontologies

- Specify which concepts appear in a certain domain and how they are related to each other
What is an ontology?

Ontologies

- Specify which concepts appear in a certain domain and how they are related to each other

Example: Simple social network

- Concepts: User, Group, Picture
- Relations:
  - Users are friends with other users
  - Users are members of groups
  - Users have pictures
  - Users are shown on pictures
Example: Ontology representing social network

User - is member of - Group
User - is friend of - Picture
User - has - Picture
User - is on - Picture
Introduction to Description Logic

- DL knowledge base consists of TBox and ABox
- Knowledge represented by concepts and roles
Introduction to Description Logic

- DL knowledge base consists of TBox and ABox
- Knowledge represented by concepts and roles

**ABox**

- `User(john)`
- `User(kate)`
- `Group(students_in_koblenz)`
- `IsFriendOf (john, kate)`
- `IsMemberOf (john, students_in_koblenz)`
Introduction to Description Logic

- DL knowledge base consists of TBox and ABox
- Knowledge represented by concepts and roles

ABox
- \text{User}(\text{john})
- \text{User}(\text{kate})
- \text{Group}(%students\_in\_koblenz)
- \text{IsFriendOf}\ ((\text{john, kate})
- \text{IsMemberOf}\ ((\text{john, students\_in\_koblenz})

TBox
- \text{User} \sqsubseteq \neg \text{Group}
Interpretations

- Defined as $I = (\Delta^I, \cdot^I)$
- Assumption: objects in $\Delta^I$ = individuals in ABox
- Assign sets of objects to concepts
- Assign sets of tuples of objects to roles
Introduction to Description Logic - Interpretations

**ABox**

*User*(john)
*User*(kate)
*Group*(students_in_koblenz)
*IsFriendOf*(john, kate)
*IsMemberOf*(john, students_in_koblenz)

**TBox**

*User* ⊑ ¬*Group*
Introduction to Description Logic - Interpretations

**ABox**

*User*(john)
*User*(kate)
*Group*(students_in_koblenz)
*IsFriendOf*(john, kate)
*IsMemberOf*(john, students_in_koblenz)

**TBox**

*User* $\sqsubseteq \neg*Group*

**Possible interpretation**

$\Delta^I = \{john, kate, students_in_koblenz\}$

$User^I = \{john, kate\}$

$Group^I = \{students_in_koblenz\}$

*IsFriendOf* $^I = \{(john, kate)\}$

*IsMemberOf* $^I = \{(john, students_in_koblenz)\}$
Queries in Description Logic

- Queries represented by Horn clauses
- Predicates represent concepts and roles
Queries in Description Logic

- Queries represented by Horn clauses
- Predicates represent concepts and roles

Example
Query: \( q(x) \leftarrow IsFriendOf(john, x) \)
Queries in Description Logic

- Queries represented by Horn clauses
- Predicates represent concepts and roles

**Example**

Query: \( q(x) \leftarrow \text{IsFriendOf}(john, x) \)
Remember: \( \text{IsFriendOf}^I = \{(john, kate)\} \)
Queries in Description Logic

- Queries represented by Horn clauses
- Predicates represent concepts and roles

Example

Query: \( q(x) \leftarrow \text{IsFriendOf}(john, x) \)

Remember: \( \text{IsFriendOf}^I = \{(john, kate)\} \)

Answer: \( q^I = \{(kate)\} \)
Queries in Description Logic

- Queries represented by Horn clauses
- Predicates represent concepts and roles

Example
Query: \( q(x) \leftarrow \text{IsFriendOf}(\text{john}, x) \)
Remember: \( \text{IsFriendOf}^I = \{(\text{john}, \text{kate})\} \)
Answer: \( q^I = \{(\text{kate})\} \)

- Certain answer to \( q \) w.r.t. \( W = \{l_1, \ldots, l_n\} \): Tuple that is in every \( q^{l_i} \)
- Shorthand notation for set of certain answers: \( \text{cert}(q, W) \)
Queries in Description Logic

- Queries represented by Horn clauses
- Predicates represent concepts and roles

Example

Query: \( q(x) \leftarrow IsFriendOf(john, x) \)
Remember: \( IsFriendOf^I = \{(john, kate)\} \)
Answer: \( q^I = \{(kate)\} \)

- Certain answer to \( q \) w.r.t. \( W = \{l_1, ..., l_n\} \): Tuple that is in every \( q^{l_i} \)
- Shorthand notation for set of certain answers: \( \text{cert}(q, W) \)
- Views are represented by queries
  - \( v(x) \leftarrow IsFriendOf(john, x) \) is view that allows user to retrieve all of john’s friends
View-based query answering

- Given: DL knowledge base $K$
- $MOD(K) = \text{set of all models of } K$
- $T = \text{TBox of } K$
- $V = \langle v_1, \ldots, v_n \rangle$: views, e.g. $v_1(x) \leftarrow \text{IsFriendOf}(john, x)$
- $E = \langle e_1, \ldots, e_n \rangle$: certain extension of $V$
- $e_i = \text{cert}(v_i, MOD(K))$, e.g. $e_1 = \{(kate)\}$
View-based query answering

- Given: DL knowledge base $K$
- $MOD(K) = \text{set of all models of } K$
- $T = \text{TBox of } K$
- $V = \langle v_1, \ldots, v_n \rangle: \text{views, e.g. } v_1(x) \leftarrow \text{IsFriendOf}(john, x)$
- $E = \langle e_1, \ldots, e_n \rangle: \text{certain extension of } V$
- $e_i = \text{cert}(v_i, MOD(K)), \text{e.g. } e_1 = \{(kate)\}$

**Solution**

Set $W$ of interpretations is solution to $(T, V, E)$ if:

- All interpretations in $W$ are models for $T$ (not necessarily for $K$!)
- $\text{cert}(v_i, W) = e_i$

Tuple $a$ is valid view-based answer to $q$ w.r.t. $(T, V, E)$ if $a \in \text{cert}(q, W)$ for all solutions $W$
Practical Example: A Simple Social Network

Concepts

- User
- Group
- Picture

Roles

- IsFriendOf
- IsMemberOf
- HasPicture
- IsOnPicture

We’ll refer to our complete knowledge base (TBox $T + ABox A$) as $K$
Practical Example: A Simple Social Network

TBox

\[ \text{Group} \sqsubseteq \neg \text{Picture} \]
\[ \text{User} \sqsubseteq \neg \text{Picture} \]
\[ \text{User} \sqsubseteq \neg \text{Group} \]

- Group, User, and Picture are disjoint
Practical Example: A Simple Social Network

ABox

User(john)
User(steve)
User(kate)

Group(students_in_koblenz)

Picture(pic1_jpg)
Picture(pic2_jpg)

IsMemberOf(john, students_in_koblenz)
IsMemberOf(kate, students_in_koblenz)

HasPicture(john, pic1_jpg)
HasPicture(john, pic2_jpg)

IsFriendOf(john, kate)

IsOnPicture(steve, pic1_jpg)
Practical Example: A Simple Social Network

Privacy policy

Users cannot see members of a group they’re not a member of:

\[ v_1(x, y) \leftarrow \text{User}($id$), \text{IsMemberOf}($id, x$), \text{IsMemberOf}(y, x) \]
Practical Example: A Simple Social Network

Privacy policy

- Users cannot see members of a group they’re not a member of:
  \[ v_1(x, y) \leftarrow \text{User}(id), \text{IsMemberOf}(id, x), \text{IsMemberOf}(y, x) \]

- Users can only see pictures of friends and pictures they are on:
  \[ v_2(x, y) \leftarrow \text{User}(id), \text{HasPicture}(x, y), \text{IsFriendOf}(id, x) \]
  \[ v_2(x, y) \leftarrow \text{User}(id), \text{HasPicture}(x, y), \text{IsOnPicture}(id, y) \]
Practical Example: A Simple Social Network

Privacy policy

- Users cannot see members of a group they’re not a member of:

\[
v_1(x, y) \leftarrow \text{User}(\text{id}), \text{IsMemberOf}(\text{id}, x), \text{IsMemberOf}(y, x)
\]

- Users can only see pictures of friends and pictures they are on:

\[
v_2(x, y) \leftarrow \text{User}(\text{id}), \text{HasPicture}(x, y), \text{IsFriendOf}(\text{id}, x)
v_2(x, y) \leftarrow \text{User}(\text{id}), \text{HasPicture}(x, y), \text{IsOnPicture}(\text{id}, y)
\]

Instantiated views for steve

\[
v_1(x, y) \leftarrow \text{User}(\text{steve}), \text{IsMemberOf}(\text{steve}, x), \text{IsMemberOf}(y, x)
v_2(x, y) \leftarrow \text{User}(\text{steve}), \text{HasPicture}(x, y), \text{IsFriendOf}(\text{steve}, x)
v_2(x, y) \leftarrow \text{User}(\text{steve}), \text{HasPicture}(x, y), \text{IsOnPicture}(\text{steve}, y)
\]
A useful interpretation

Consider the following interpretation $I$:

\[ \triangle I = \{ john, steve, kate, students\_in\_koblenz, pic1\_jpg, pic2\_jpg \} \]

User $I = \{ john, steve, kate \}$

Group $I = \{ students\_in\_koblenz \}$

Picture $I = \{ pic1\_jpg, pic2\_jpg \}$

IsMemberOf $I =$

\[ \{( john, students\_in\_koblenz), ( kate, students\_in\_koblenz) \} \]

HasPicture $I = \{( john, pic1\_jpg), ( john, pic2\_jpg) \}$

IsFriendOf $I = \{( john, kate) \}$

IsOnPicture $I = \{( steve, pic1\_jpg) \}$

- $I$ is a model for $K$
- Every fact in $I$ is in every other model for $K$ as well
- $\text{cert}(v_i, \{ I \}) = \text{cert}(v_i, MOD(K))$
Practical Example: A Simple Social Network

Computing the certain extensions

Remember:

\[ v_1(x, y) \leftarrow User(steve), \text{IsMemberOf}(steve, x), \text{IsMemberOf}(y, x) \]
Computing the certain extensions

Remember:
\[ v_1(x, y) \leftarrow User(steve), IsMemberOf(steve, x), IsMemberOf(y, x) \]

\[ IsMemberOf(steve, x) \text{ not true in } I \text{ for any } x \]
\[ \implies \text{ body of } v_1 \text{ always false under } I \text{ (no answer)} \]
\[ \implies e_1 = \text{cert}(v_1, MOD(K)) = \text{cert}(v_1, \{I\}) = \{\} \]
Practical Example: A Simple Social Network

Computing the certain extensions

Remember:

\[ \nu_2(x, y) \leftarrow User(steve), HasPicture(x, y), IsFriendOf(steve, x) \]

\[ \nu_2(x, y) \leftarrow User(steve), HasPicture(x, y), IsOnPicture(steve, y) \]
Practical Example: A Simple Social Network

Computing the certain extensions

Remember:
\[ v_2(x, y) \leftarrow User(steve), HasPicture(x, y), IsFriendOf(steve, x) \]
\[ v_2(x, y) \leftarrow User(steve), HasPicture(x, y), IsOnPicture(steve, y) \]

- \( IsFriendOf(steve, x) \) not true in \( I \) for any \( x \)
  \[ \implies \text{first body of } v_2 \text{ always false under } I \text{ (no answer)} \]
- But: \( User(steve), HasPicture(john, pic1\_jpg) \), \( IsOnPicture(steve, pic1\_jpg) \) true in \( I \)
  \[ \implies (john, pic1\_jpg) \text{ is answer under } I \]
  \[ \implies e_2 = cert(v_2, MOD(K)) = cert(v_2, \{I\}) = \{(john, pic1\_jpg)\} \]
Practical Example: A Simple Social Network

Query entered by steve

\[ q_1(x) \leftarrow \text{IsMemberOf}(x, \text{students\_in\_koblenz}) \]
Practical Example: A Simple Social Network

Query entered by steve

\[ q_1(x) \leftarrow IsMemberOf(x, \text{students\_in\_koblenz}) \]

Remember:

\[ v_1(x, y) \leftarrow User(steve), IsMemberOf(steve, x), IsMemberOf(y, x) \]

\[ v_2(x, y) \leftarrow User(steve), HasPicture(x, y), IsFriendOf(steve, x) \]

\[ v_2(x, y) \leftarrow User(steve), HasPicture(x, y), IsOnPicture(steve, y) \]
Practical Example: A Simple Social Network

Query entered by steve

\[ q_1(x) \leftarrow \text{IsMemberOf}(x, \text{students\_in\_koblenz}) \]

Remember:

\[ v_1(x, y) \leftarrow \text{User}(\text{steve}), \text{IsMemberOf}(\text{steve}, x), \text{IsMemberOf}(y, x) \]
\[ v_2(x, y) \leftarrow \text{User}(\text{steve}), \text{HasPicture}(x, y), \text{IsFriendOf}(\text{steve}, x) \]
\[ v_2(x, y) \leftarrow \text{User}(\text{steve}), \text{HasPicture}(x, y), \text{IsOnPicture}(\text{steve}, y) \]

Note: \( \{I\} \) is solution to \((T, V, E)\)

Create \( I' \) where \( I' \) is identical to \( I \) except that \( \text{IsMemberOf}I' = \{\} \)

\[ \Rightarrow \text{cert}(v_1, \{I'\}) = e_1, \text{cert}(v_2, \{I'\}) = e_2 \]
\[ \Rightarrow \{I'\} \text{ is solution, but } \text{cert}(q_1, \{I'\}) = \{\} \]
\[ \Rightarrow q_1 \text{ has no answer} \]
Practical Example: A Simple Social Network

Query entered by steve

\[ q_2(x) \leftarrow \text{HasPicture}(john, x) \]
Practical Example: A Simple Social Network

Query entered by steve
\[ q_2(x) \leftarrow \text{HasPicture}(john, x) \]

Remember:
\[ v_2(x, y) \leftarrow \text{User}(steve), \text{HasPicture}(x, y), \text{IsFriendOf}(steve, x) \]
\[ v_2(x, y) \leftarrow \text{User}(steve), \text{HasPicture}(x, y), \text{IsOnPicture}(steve, y) \]
Practical Example: A Simple Social Network

Query entered by steve

\[ q_2(x) \leftarrow \text{HasPicture}(john, x) \]

Remember:

\[ v_2(x, y) \leftarrow \text{User}(steve), \text{HasPicture}(x, y), \text{IsFriendOf}(steve, x) \]
\[ v_2(x, y) \leftarrow \text{User}(steve), \text{HasPicture}(x, y), \text{IsOnPicture}(steve, y) \]

\((john, \text{pic1}_\text{jpg}) \in e_2\)
\[ \implies (john, \text{pic1}_\text{jpg}) \in \text{cert}(v_2, \{I\}) \]
\[ \implies \text{HasPicture}(john, \text{pic1}_\text{jpg}) \text{ true in every solution} \]
\[ \implies (\text{pic1}_\text{jpg}) \text{ is answer to } q_2. \text{ What about } (\text{pic2}_\text{jpg})? \]
Practical Example: A Simple Social Network

Query entered by steve

\[ q_2(x) \leftarrow \text{HasPicture}(john, x) \]

Remember:

\[ v_2(x, y) \leftarrow \text{User}(steve), \text{HasPicture}(x, y), \text{IsFriendOf}(steve, x) \]
\[ v_2(x, y) \leftarrow \text{User}(steve), \text{HasPicture}(x, y), \text{IsOnPicture}(steve, y) \]

\[(john, \text{pic1\_jpg}) \in e_2 \]
\[\Rightarrow (john, \text{pic1\_jpg}) \in \text{cert}(v_2, \{l\}) \]
\[\Rightarrow \text{HasPicture}(john, \text{pic1\_jpg}) \text{ true in every solution} \]
\[\Rightarrow (\text{pic1\_jpg}) \text{ is answer to } q_2. \text{ What about } (\text{pic2\_jpg})? \]

Construct \( l^* \) such that \( l^* \) is identical to \( l \) except that
\[ \text{HasPicture}^l = \{(john, \text{pic1\_jpg})\} \text{ ((john, \text{pic2\_jpg}) left out).} \]
\[ \text{cert}(v_i, \{l^*\}) = \text{cert}(v_i, \{l\}) \Rightarrow \{l^*\} \text{ is solution} \]
\[ (\text{pic2\_jpg}) \notin \text{cert}(q_2, \{l^*\}) \Rightarrow \text{not an answer} \]
Conclusion

What have we accomplished?

- Explained why access control in knowledge bases is necessary
- Shown how ontologies can be written in Description Logics
- Transferred concept of views from relational database world to logic domain
- Shown how views can be used to control access to DL ontologies

Prospect for the future

- Increase in popularity of DL as ontology language?
- Complex DL ontologies with advanced access control mechanisms?
Conclusion

What have we accomplished?

- Explained why access control in knowledge bases is necessary
- Shown how ontologies can be written in Description Logics
- Transferred concept of views from relational database world to logic domain
- Shown how views can be used to control access to DL ontologies

Prospect for the future

- Increase in popularity of DL as ontology language?
- Complex DL ontologies with advanced access control mechanisms?